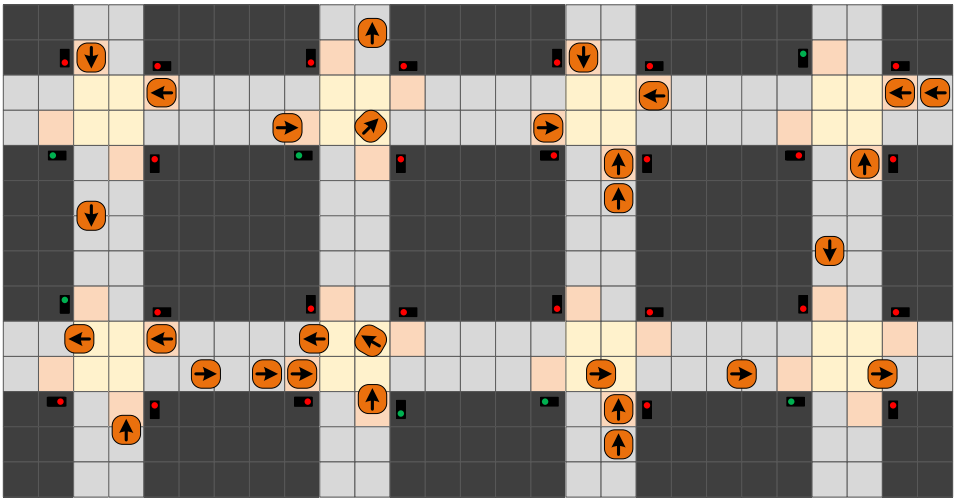
LEHRVERANSTALTUNG – JAHR/SEMESTER

Aufgabe 1: TITEL DER AUFGABE

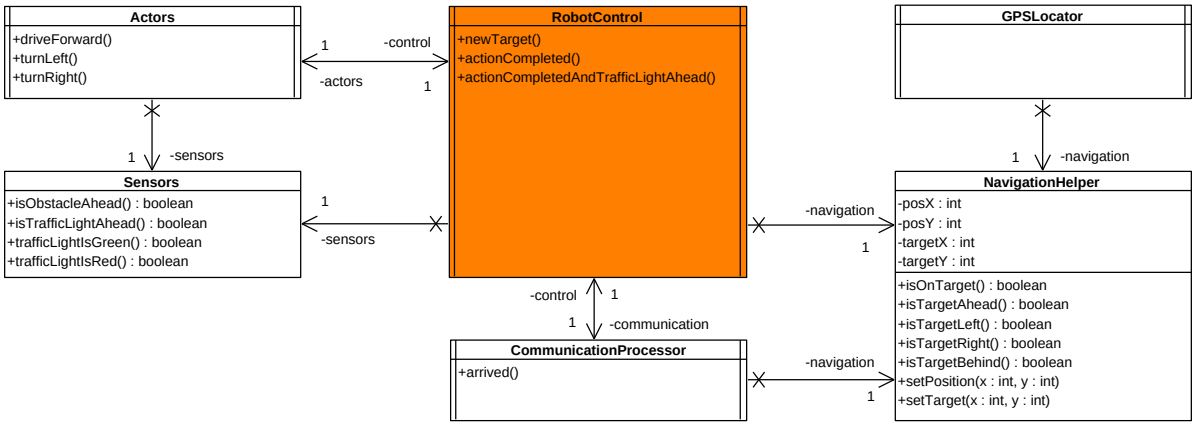
1 Entwurf „Roboter im Straßenverkehr“ *(46 Punkte)*

Gegeben ist ein Straßenverkehrssystem mit einem rasterförmigen Netz von horizontalen und vertikalen Straßen. An jeder Kreuzung befinden sich Ampeln, die den Verkehrsfluss steuern.



In diesem auf einzelne Felder aufgeteilten, zweidimensionalen Straßennetz fahren autonome Roboter zu von einem zentralen Server vorgegebenen Zielen. Die Roboter fahren im Rechtsverkehr (d.h. immer auf der in Fahrtrichtung gesehen rechten Spur) und achten beim Fahren autonom darauf, dass sie 1. nicht auf andere Roboter auffahren, 2. die Ampelzeichen beachten und 3. an Kreuzungen passend abbiegen.

In dieser Aufgabe soll die Steuerungssoftware der Roboter, konkret die aktive Klasse `RobotControl`, implementiert werden. Für die Software der Roboter ist folgendes Klassendiagramm gegeben:



Ein Roboter ist zu Beginn pausiert. Im weiteren Verlauf ist er immer entweder pausiert oder hat ein bestimmtes Fahrziel, das ihm vom zentralen Server zugewiesen wird und zu dem er fährt. Wenn ein neues Ziel beim `CommunicationProcessor`-Objekt ankommt, setzt dieses mit der Methode `setTarget(x,y)` den `NavigationHelper` über die neuen Koordinaten in Kenntnis und informiert das `RobotControl`-Objekt mit dem Methodenaufruf `newTarget()`. Es kann angenommen werden, dass weder die Startposition des Roboters bei Erhalt eines Auftrags noch dessen Ziel jemals auf einer Kreuzung oder auf einem Feld direkt vor oder direkt hinter einer Kreuzung liegt. Außerdem ist das Ziel niemals gleich der Startposition.

Wenn ein Roboter ein Ziel hat fährt er, solange das Ziel noch nicht erreicht ist, Feld für Feld geradeaus entlang der Straßen, und nutzt ggf. die Kreuzungen, um nach links oder rechts abzubiegen. Nach jeder Aktion, bei der er potentiell das Ziel erreicht haben könnte, wird dies mit der Abfrage `isOnTarget()` beim `NavigationHelper` überprüft.

Um Kollisionen zu vermeiden ruft das `RobotControl`-Objekt vor jeder Aktion, bei der es notwendig ist, die Methode `isObstacleAhead()` beim `Sensors`-Objekt auf. Wenn die Methode `true` zurückgibt bedeutet das, dass das nächste Feld in Fahrtrichtung bereits mit einem Roboter belegt ist. Das `RobotControl`-Objekt muss dann solange immer wieder die Sensoren abfragen, bis das Feld frei geworden ist.

Eine Aktion eines Roboters ist immer entweder die Bewegung um ein Feld vorwärts in Fahrtrichtung, das Drehen auf der Stelle nach links oder das Drehen auf der Stelle nach rechts. Diese Aktionen werden vom `RobotControl`-Objekt jeweils mit den Methoden `driveForward()`, `turnLeft()` und `turnRight()` beim `Actors`-Objekt aufgerufen. Wenn die entsprechende Aktion abgeschlossen ist, meldet das `Actors`-Objekt dies mit dem Aufruf `actionCompleted()` an das `RobotControl`-Objekt zurück, welches dann die nächste Aktion aufrufen kann.

Das `Actors`-Objekt ruft zudem nach jedem Vorwärtsfahren die Methode `isTrafficLightAhead()` beim `Sensors`-Objekt auf, um festzustellen, ob die Warteposition vor einer Ampel erreicht wurde. Wenn dies der Fall ist, wird beim `RobotControl`-Objekt statt `actionCompleted()` die Methode `actionCompletedAndTrafficLightAhead()` aufgerufen.

Wenn ein Roboter vor einer Ampel steht muss er warten, bis diese grün ist. Dazu wird wiederholt `trafficLightIsGreen()` oder `trafficLightIsRed()` beim `Sensors`-Objekt abgefragt. Sobald die Ampel grün zeigt kann die Kreuzung befahren werden und geradeaus, nach links

oder nach rechts verlassen werden, wobei das Durchfahren der Kreuzung immer aus einer Sequenz von mehreren, beim `Actors`-Objekt aufgerufenen Einzelaktionen besteht. Es kann davon ausgegangen werden, dass die Ampel nur Roboter aus einer Richtung einfahren lässt, sodass links abbiegende Roboter kein Problem mit Gegenverkehr haben.

Um festzustellen, in welche Richtung der Roboter auf der Kreuzung fahren muss, stellt das `NavigationHelper`-Objekt die Methoden `isTargetAhead()`, `isTargetLeft()` und `isTargetRight()` bereit, die jeweils einen boolean zurückgeben. Die Methoden funktionieren zuverlässig sowohl bei einer Roboter-Position vor als auch auf einer Kreuzung. Es ist möglich, dass mehrere dieser Methoden in einer Situation `true` ergeben.

Wenn der Roboter sein Ziel erreicht hat pausiert er sofort und meldet die Ankunft mit dem Aufruf `arrived()` an den `CommunicationProcessor`. Das `RobotControl`-Objekt pausiert bis es einen neuen `newTarget()`-Aufruf erhält.

a) Modellieren Sie die Struktur der beschriebenen Robotersteuerungssoftware für *einen* Roboter mit einem Objektdiagramm. Der Roboters ist an Position `x=8`, `y=14`, das Ziel ist `x=10`, `y=16`. Achten Sie auf Konsistenz zum vorgegebenen Klassendiagramm. *(8 Punkte)*

b) Modellieren Sie das Verhalten der aktiven Klasse `RobotControl` mit einem Zustandsdiagramm. Stellen Sie insbesondere sicher, dass Ihr Zustandsdiagramm ein sicheres Fahrverhalten realisiert, sodass die Roboter 1. nicht auf andere Roboter auffahren, 2. die Ampelzeichen beachten und 3. an Kreuzungen passend abbiegen.

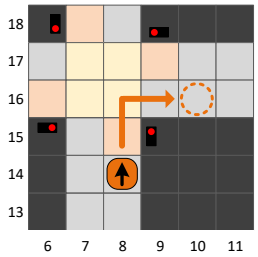
   In Situationen, in denen regelmäßig eine Abfrage erfolgen muss (z.B. beim Warten auf eine grüne Ampel), verwenden Sie `after 100ms` als Trigger und nicht den Trigger `when`.

   Achten Sie auf Konsistenz zum vorgegebenen Klassendiagramm. *(24 Punkte)*

   > **Hinweis:** Nutzen Sie für diese Teilaufgabe die YAKINDU Statechart Tools für die Modellierung. Laden Sie bei der Abgabe neben der üblichen PDF-Datei ein `.zip`-Archiv im Moodle hoch, welches die `simpletraffic_robot.ysc`-Datei mit Ihren Diagramm enthält.
   >
   > Wir stellen Ihnen für diese Übung ein Framework zur Verfügung, mit dem Sie das durch ihr Zustandsdiagramm definierte Fahrverhalten der Roboter selbst erproben und beobachten können. Weitere Informationen zur Installation und Verwendung der YAKINDU Statechart Tools und des Simulators finden Sie in dem im Moodle hinterlegten Dokument `Toolhinweise.pdf`.
   >
   > Beachten Sie, dass die Diagramm-Syntax in den YAKINDU Statechart Tools von der üblichen UML-Syntax abweicht. Die wichtigsten Besonderheiten sind ebenfalls in `Toolhinweise.pdf` kurz aufgelistet. Für die Klausur müssen Sie nur die Syntax von UML-Zustandsdiagrammen beherrschen.

c) Modellieren Sie die Interaktion zwischen den Objekten der Robotersteuerungssoftware bei dem folgenden beispielhaften Ablauf mit einem Sequenzdiagramm:

Der Roboter ist initial pausiert. Die initiale Position ist `x=8`, `y=14`. Der `CommunicationProcessor` erhält vom zentralen Server den Auftrag, zur Position `x=10`, `y=16` zu fahren und gibt diesen Befehl entsprechend weiter. Es werden keine anderen Roboter oder sonstige Hindernisse auf der Strecke angetroffen. Beim Erreichen der Ampel ist diese bei der ersten Abfrage der Sensoren rot, aber bereits bei der zweiten Abfrage grün. Nach Ankunft am Ziel meldet der Roboter dies über den `CommunicationProcessor` an den zentralen Server. Es folgen keine weiteren Fahrbefehle. Der Ablauf ist damit beendet.

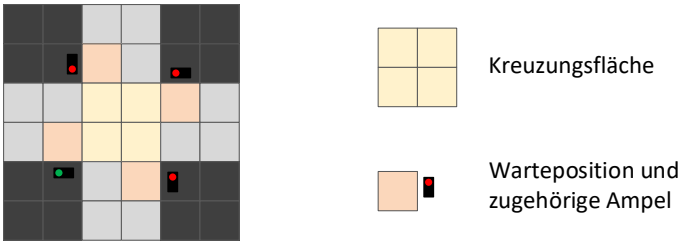


Modellieren Sie nur die Interaktion der Objekte der Klassen `RobotControl`, `Actors`, `Sensors`, `CommunicationProcessor` und `NavigationHelper`. Das Objekt der Klasse `GPSLocator` sowie der zentrale Server müssen nicht beachtet werden.

Verwenden Sie in den Aufrufen konkrete Werte als Parameter oder Rückgabewert. Achten Sie bei der Modellierung auf Konsistenz zum vorgegebenen Klassendiagramm und zu Ihrem UML-Zustandsdiagramm aus Teilaufgabe b). *(14 Punkte)*

2 Entwurf „Ampel im Straßenverkehr“ *(25+15 Punkte)*

In dieser Aufgabe wird eine Steuerungssoftware für Ampeln betrachtet, die in dem in Aufgabe 1 genannten Verkehrssystem eingesetzt werden soll.

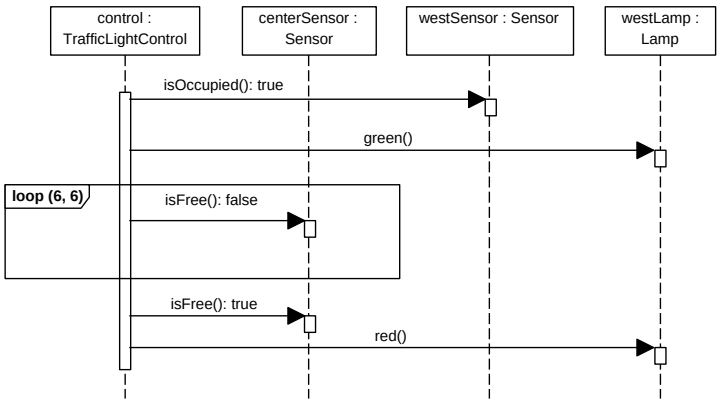


Die Steuerungssoftware wird durch eine aktive Klasse `TrafficLightControl` umgesetzt. Ein `TrafficLightControl`-Objekt ist immer für die Verkehrssteuerung auf einer kompletten Kreuzung zuständig.

Die Steuerung des Verkehrsflusses erfolgt durch das Wechseln der angezeigten Farbe bei den jeweils vier zugehörigen Ampeln, die in die Himmelsrichtungen Nord, Ost, Süd und West zeigen. Für jede der Ampeln an einer Kreuzung steht dem `TrafficLightControl`-Objekt ein `Lamp`-Objekt zur Steuerung zur Verfügung. Eine Ampel kann immer nur entweder rot oder grün anzeigen.

Zusätzlich verfügt die Steuerungssoftware über fünf Sensoren, die feststellen können, ob sich in einem Bereich Roboter aufhalten oder nicht. Ein Sensor beobachtet die zentrale Fläche der Kreuzung, außerdem gibt es für jede Himmelsrichtung einen weiteren Sensor, der die Warteposition vor den Ampeln beobachtet. Jeder der Sensoren kann über ein `Sensor`-Objekt angesteuert werden.

a) Modellieren Sie ein Klassendiagramm für die Steuerungsoftware der Ampel. Neben dem obenstehenden Text ist dazu die folgende beispielhafte Sequenz von Interaktionen vorgegeben:



Achten Sie auf Konsistenz zum Sequenzdiagramm. Nehmen Sie an, dass die Bestand-

teile der Ampelsteuerung für alle vier Himmelsrichtungen äquivalent funktionieren und benannt sind, auch wenn nicht alle Richtungen im Sequenzdiagramm vorkommen. *(7 Punkte)*

b) Modellieren Sie das Verhalten der aktiven Klasse `TrafficLightControl` mit einem UML-Zustandsdiagramm, sodass folgende Eigenschaften erfüllt werden:

- Die Ampeln in den einzelnen Himmelsrichtungen werden reihum einzeln auf grün geschaltet, die übrigen drei Ampeln sind währenddessen rot.
- Eine Ampel wird erst dann auf grün gestellt, wenn sich auf der Kreuzungsfläche keine Roboter mehr befinden. Im pessimistischsten Fall kann angenommen werden, dass eine Durchquerung einer Kreuzung für einen Roboter nicht länger als 3 Sekunden dauert.
- Zeiten, in denen alle vier Ampeln gleichzeitig rot zeigen, sollen möglichst kurz gehalten werden (zumindest sofern auf einer Warteposition noch ein Roboter wartet).
- Kein Roboter, der die Warteposition direkt vor der Ampel erreicht, soll auf dieser länger als 30 Sekunden warten müssen.

In Situationen, wo regelmäßig eine Abfrage erfolgen muss (z.B. bei Warten bis die Kreuzungsfläche frei ist), verwenden Sie `after 100ms` als Trigger und nicht den Trigger `when`.

Achten Sie bei der Modellierung auf Konsistenz zu Ihrem Klassendiagramm. Der beispielhafte Ablauf im bei Teilaufgabe a) gegebenen Sequenzdiagramm muss nicht beachtet werden. *(18 Punkte)*

> **Hinweis:** Nutzen Sie für diese und die folgende Teilaufgabe die YAKINDU Statechart Tools für die Modellierung. Laden Sie bei der Abgabe neben der üblichen PDF-Datei ein `.zip`-Archiv im Moodle hoch, welches die `simpletraffic_light.ysc`-Datei mit Ihren Diagramm enthält.
>
> Wir stellen Ihnen für diese Übung ein Framework zur Verfügung, mit dem Sie das durch ihr Zustandsdiagramm definierte Fahrverhalten der Roboter selbst erproben und beobachten können. Weitere Informationen zur Installation und Verwendung der YAKINDU Statechart Tools und des Simulators finden Sie in dem im Moodle hinterlegten Dokument `Toolhinweise.pdf`.
>
> Beachten Sie, dass die Diagramm-Syntax in den YAKINDU Statechart Tools von der üblichen UML-Syntax abweicht. Die wichtigsten Besonderheiten sind ebenfalls in `Toolhinweise.pdf` kurz aufgelistet. Für die Klausur müssen Sie nur die Syntax von UML-Zustandsdiagrammen beherrschen.

c) **Bonusaufgabe:** Verbessern Sie Ihre Lösung aus Teilaufgabe b), sodass der Roboterverkehr insgesamt effizienter wird (und die Roboter im Durchschnitt schneller ihre Ziele erreichen). Mögliche Ansätze dafür:

- Eine Ampel schaltet in einer Himmelsrichtung nur dann auf grün, wenn bereits ein

Roboter auf der Warteposition steht.

- Ampelphasen werden verkürzt falls keine Roboter auf die Warteposition nachrücken.
- Es wird beachtet, in welche Richtung die Roboter abbiegen wollen, um ggf. mehrere Ampeln gleichzeitig auf grün schalten zu können. Dazu kann angenommen werden, dass die `Sensor`-Objekte die zusätzliche Methode `getDirection()` zur Verfügung stellt, die `Direction.LEFT` oder `Direction.RIGHT` zurückgibt, wenn der wartende Roboter abbiegen will, und `Direction.AHEAD`, wenn er geradeausfahren will oder wenn kein Roboter wartet. Diese Methode muss nicht in Teilaufgabe a berücksichtigt werden.

Sie müssen kein zusätzliches Diagramm angeben, es genügt wenn Sie Ihre Abgabe aus Teilaufgabe b) ergänzen. *(0+15 Punkte)*

Für diese Aufgabe gibt es keine regulären Punkte, aber bis zu 15 Bonuspunkte, mit denen Sie andere Punkte, die Ihnen in dieser Übung abgezogen werden, ausgleichen können.