OPPA European Social Fund
Prague & EU: We invest in your future.

In this tutorial we assume the instantiation of a car agent by extension of
`tacv.universe.entity.car.PahCarAgent`

1. **Sensing and data access.**

   PahCarAgent (derived from AbstractCarAgent) has registered basic car sensor allowing to obtain actual car position, speed, and distance to actual waypoint (note the world dimension is 1500x1500).

   ```
   Point carSensor.senseMyPosition();

   float carSensor.senseSpeed();

   Double carSensor.senseDistanceToWaypoint();
   ```

   Recommended car speed is accessible as AbstractCarAgent.TARGET_SPEED.

   There is also registered alarm sensor enabling time related operations available. It can be used by following code:

   ```
   alarmSensor.registerAlarmCallback(long deltaTimeInMillis, new AlarmCallback() {
       @Override
       public void senseAlarm(long simulationTime) {
           // put your code here – it will be invoked after
           // deltaTimeInMillis of simulated time
       }
   });
   ```

2. **Driving the car.**

   The car can be driven using GoToWaypointCarActuator, which is able to regulate the steering and engine force to reach desired waypoint. To use the actuator, simply put following code to the car agent class constructor:

   ```
   GoToWaypointCarActuator goToWaypointCarActuator =
   this.handler.addAction(GoToWaypointCarActuator.class, this);
   ```

   Then the control of the vehicle can be done using methods:

   ```
   goToWaypointCarActuator.actGoToWaypoint(Point targetWaypoint, Float targetSpeed,
                                           boolean isForward);

   goToWaypointCarActuator.actHalt();
   ```

   Achieving the targetWaypoint can be tested using waypoint reached callback. Put following code to the car agent class constructor:

   ```
   carSensor.registerWaypointReachedCallback(new WaypointReachedCallback() {
       @Override
       public void waypointReached(Point wayPoint) {
           // put your code here – it will be invoked when
           // the car reaches the waypoint
       }
   });
   ```

   Visualization of the path-plan in 3D can be done by:

   ```
   goToWaypointCarActuator.setVisualPlan(VisualPlanLayer.VisualPlan visualPlan);
   ```

3. Handling the environment.

The environment handler (class AgentScoutEnvironment.AgentScoutEnvironmentHandler), accessible as AbstractCarAgent.handler allows the agent to access the environment. The most important methods are:

`Random handler.getRandom();` for accessing random generator to ensure simulation repeatability

```
UrbanMap handler.getMap(); for accessing map structures
```

```
Terrain handler.getTerrain(); for accessing terrain structures
```

```
CollisionZone handler.getCollisionZone(); for accessing collision zones
```

```
float handler.getAltitude(float x, float y); to get an altitude of the terrain
```

The structures containing the street graph and buildings information are accessible by methods of UrbanMap.

```
List<Building> handler.getMap().getBuildings();
```

```
StreetGraph handler.getMap().getStreetGraph();
```

Collisions with the objects in the environment can be tested using (in case of problems with collision tests, set z-axis to small positive number for collision tests, i.e. 5 – it is because the collision zones are projected to z=0 with height of 1000; **!beware of mutability of vecmath – clone the variable before setting anything into its internal variables if you don't mean to change the source!**):

```
boolean handler.getCollisionZone().testLine(Point3d point1, Point3d point2);
```

```
boolean handler.getCollisionZone().testPoint(Point3d point);
```

OPPA European Social Fund
Prague & EU: We invest in your future.