Computer Assignment 6: SimpleMover and MoverManagers

While the SimplestMover and corresponding Mover Managers developed in class are a useful first step, in this assignment you will take the next step.

# SimpleMover

The SimpleMover class is a small extension of the functionality of the SimplestMover. It has the same parameters and state variables, but its Event Graph is slightly more complicated (see Figure 1).

## Parameters

* xI –initialLocation [Point2D]
* s – maxSpeed [double]

## State Variables

* x0 – lastStopLocation (xI) [Point2D]
* v – velocity (0) [Point2D]
* t0 – startMoveTime (0.0) [double]
* d – destination (NaP)[[1]](#footnote-1) [Point2D]
* x0 + (simTime – t0) v – currentLocation: an “implicit” state variable. [[2]](#footnote-2)

## Event Graph



Figure . Simple Mover Event Graph

## Implementation Notes

* Create a package called mv3302.mover; call your class SimpleMover
* The SimpleMover class should extend SimEntityBase and implement the simkit.smd.Mover interface.[[3]](#footnote-3)
* The simkit.smd.Mover interface will require a number of methods, to which you will need to add some additional ones based on the description above.
* Note the event OrderStop – this will be used for interacting with a MoverManager.
* The argument (m) in StartMove, EndMove, and Stop events is so that listeners can identify which Mover it heard. Note that MoveTo and OrderStop events schedule their events with the argument this. That ensures that listeners “hearing” those events can identify the particular SimpleMover
* In particular, the Mover interface requires methods doStartMove(Mover) and doStop(Mover).
* In the doStartMove() method you can either define a local variable tM (time to move) in the StartMove event or simply use the value ||d-x0||/s (computed in Java of course) in the waitDelay statement.
* For the (implicit) state variable currentLocation, write a setter that returns a Point2D whose coordinates are given by the equation of motion, x0 + (simTime – t0) v0.
* The default toString() that lists the parameters is not as useful as one that lists the current “state” – the states of interest are currentLocation and the velocity. Override toString() to return the SimpleMover’s name, currentLocation, and velocity in a single line.
* Add a method called paramString() that returns the super.toString(). This is to verify the parameters being passed.
* To test the movement events, schedule a MoveTo event after Schedule.reset() but before Schedule.startSimulation() and confirm that an instance of SimpleMover arrives at the correct location at the correct time. To do this, invoke waitDelay on the SimpleMover object.
* To test the OrderStop and Stop events, starting with the previous test, schedule an OrderStop event at a time prior to when your SimpleMover arrives at its destination. As with the previous test, do this after Schedule.reset() and before Schedule.startSimulation(). Use verbose mode of Schedule and a SimplePropertyDumper for this.

# SimplePathMoverManager

The SimplePathMoverManager extends the functionality of the SimplestPathMoverManager from class (see Figure 2).

The first difference is the signature of EndMove, which must match that of the one in SimpleMover in order to respond to it. The second is that when the last waypoint is reached, EndMove schedules Stop, which in turn schedules OrderStop. OrderStop in turn will be heard by the SimpleMover, which will cause it to schedule its Stop(Mover) event. The reason for this sequence is that the Stop(Mover) event in SimpleMover will need to be “heard” by other components (to be developed later).

## Parameters

* path –Point2D[] array
* startOnRun – true if NextWaypoint is event scheduled by Run; false otherwise

## State Variable

* nextIndex – index of next Point2D in path (-1)
* nextWaypoint – next waypoint, a Point2D with coordinates (NaN, NaN)

## SimplePathMoverManager Event Graph



Figure . SimplePathMoverManager Event Graph

## Listener Diagram

The SimpleMover and MoverManager instances should be SimEventListeners to each other, as shown below. The connections can be made in the main method.



## Implementation Notes

* Put your SimplePathMoverManager class in the mv3302.mover package along with SimpleMover.
* Be sure to use clone() in the setters & getters for the path parameter;
* Note that the mover will hear the Stop event but won’t respond due to the difference in arguments. The OrderStop event however will be heard and respond appropriately.
* It should not be necessary to override the default toString() for SimplePathMoverManager.
* To test your SimplePathMoverManager, write a class with a main method in the mv3302.mover.run package called Assignment6. Your test should instantiate a SimpleMover and a SimplePathMoverManager and confirm that the waypoints in path are being met at the correct times.
* Use the following data:
* SimpleMover: initialLocation at (0.0, 250.0), max speed = 30.0
* SimpleMoverManager path: (-200.0, 0.0), (-200.0, 250.0), (200.0, 250.0), (0.0, 250.0)

# SimplePatrolMoverManager

The SimplePatrolMoverManager has identical parameters and state variables as SimplePathMoverManager; the only difference is that SimplePatrolMoverManager returns to the starting waypoint at the last waypoint instead of stopping, as the SimplePathMoverManager does. Therefore, it can subclass SimplePathMoverManager and override the EndMove event, as shown below.

## SimplePatrolMoverManager Event Graph



Figure . SimplePatrolMoverManager Event Graph

## Implementation Notes

* The heavy circle for EndMove and NextWP events in SimplePatrolMoverManager indicates that the superclass doEndMove method is *not* invoked (unlike what you have been doing with subclasses of ArrivalProcess), but completely overridden.
* The listener relationship with its Mover should be the same as that of SimplePathMoverManager.
* In the Assignment6 class main method, add the following for a second SimpleMover with a PatrolMoverManager:
* SimpleMover: initial location = (0.0, 150.0), max speed = 40.0
* SimplePatrolMoverManager: path = (0.0, 300.0), (0.0, -100.0)
* Since the SimplePatrolMoverManager doesn’t have a natural stopping place (line SimplePathMoverManager), forcibly stop it a time 10.0. This will require the following in your main class:

Schedule.reset();

simplePatrolMoverManager.waitDelay("Stop", 10.0);

Schedule.startSimulation();

* Note that the SimplePathMoverManager will continue running after the SimplePatrolMoverManager stops.

# SimpleRandomMoverManager

This mover manager differs from the previous two in that it does not have any state. Rather, its movement logic is to generate a random location and send its Mover there. When its Mover arrives, another random location is generated, and the Mover dispatched to it. This continues until “told” to Stop (i.e., the Stop event is scheduled).

## Parameters

* gen – coordinateGenerator: RandomVariate[] array of length 2, used to generate the random coordinates for the next waypoint
* startOnRun – true if Run scheduled Start (as in previous).

## SimpleRandomMoverManager Event Graph



Figure . SimpleRandomMoverManager Event Graph

## Implementation Notes

* Instantiate a SimpleRandomMoverManager in the same main method as the first two. Use the following parameters:
* coordinateGenerator: {Uniform(-250, 250), Uniform(-100, 300)}
* startOnRun = true
* Use a SimpleMover located at the origin with a maximum speed of 50.0.
* Be sure to use clone() for the setter/getter for coordinateGenerator.
* Add these to the main method in Assignment6.
* To avoid an infinite run, schedule a Stop event on the instance of SimpleRandomMoverManager with a delay sufficient to demonstrate the movement. As before, this should be after Schedule.reset() and before Schedule.startSimulation(). Schedule the Stop event for the SimpleRandomMoverManager at time 20.0.
* The listener relationship with its Mover should be the same as that of SimplePathMoverManager.

# Deliverables

Push your code to Gitlab by the due date. Note: Sample output is provided as an attachment to the assignment in Sakai.

1. Recall that NaP is a Point2D with coordinates (NaN, NaN) [↑](#footnote-ref-1)
2. currentLocation is not a DES state variable – this expression is what is returned by getCurrentLocation() [↑](#footnote-ref-2)
3. Do *not* implement the simkit.smdx.Mover interface! [↑](#footnote-ref-3)