import java.util.HashMap;

import java.util.Map;

/\*\*

\* This class stores the basic state necessary for the A\* algorithm to compute a

\* path across a map. This state includes a collection of "open waypoints" and

\* another collection of "closed waypoints." In addition, this class provides

\* the basic operations that the A\* pathfinding algorithm needs to perform its

\* processing.

\*\*/

public class AStarState

{

/\*\* This is a reference to the map that the A\* algorithm is navigating. \*\*/

private Map2D map;

private HashMap<Location, Waypoint> opened = new HashMap<>();

private HashMap<Location, Waypoint> closed = new HashMap<>();

/\*\*

\* Initialize a new state object for the A\* pathfinding algorithm to use.

\*\*/

public AStarState(Map2D map)

{

if (map == null)

throw new NullPointerException("map cannot be null");

this.map = map;

}

/\*\* Returns the map that the A\* pathfinder is navigating. \*\*/

public Map2D getMap()

{

return map;

}

/\*\*

\* This method scans through all open waypoints, and returns the waypoint

\* with the minimum total cost. If there are no open waypoints, this method

\* returns <code>null</code>.

\*\*/

public Waypoint getMinOpenWaypoint()

{

Waypoint min = null;

for (Map.Entry<Location, Waypoint> entry : opened.entrySet()) {

if (min == null) {

min = entry.getValue();

} else {

if (min.getTotalCost() > entry.getValue().getTotalCost() ) {

min = entry.getValue();

}

}

}

return min;

}

/\*\*

\* This method adds a waypoint to (or potentially updates a waypoint already

\* in) the "open waypoints" collection. If there is not already an open

\* waypoint at the new waypoint's location then the new waypoint is simply

\* added to the collection. However, if there is already a waypoint at the

\* new waypoint's location, the new waypoint replaces the old one <em>only

\* if</em> the new waypoint's "previous cost" value is less than the current

\* waypoint's "previous cost" value.

\*\*/

public boolean addOpenWaypoint(Waypoint newWP)

{

if (!opened.containsKey(newWP.getLocation())) {

opened.put(newWP.getLocation(), newWP);

return true;

} else {

Location location = newWP.getLocation();

for (Map.Entry<Location, Waypoint> entry : opened.entrySet()) {

if (entry.getKey().equals(location) && entry.getValue().getRemainingCost() > newWP.getRemainingCost()) {

opened.put(location, newWP);

return true;

} else {

return false;

}

}

}

return false;

}

/\*\* Returns the current number of open waypoints. \*\*/

public int numOpenWaypoints()

{

return opened.size();

}

/\*\*

\* This method moves the waypoint at the specified location from the

\* open list to the closed list.

\*\*/

public void closeWaypoint(Location loc)

{

closed.put(loc, opened.remove(loc));

}

/\*\*

\* Returns true if the collection of closed waypoints contains a waypoint

\* for the specified location.

\*\*/

public boolean isLocationClosed(Location loc)

{

return closed.containsKey(loc);

}

}