Solution: Turn Right

created by Lisa Wang

```
/* Function: findCheapestPath
 * Usage: int cost = findCheapestPath(start, end, curOrientation, path, world, markers);
 * ------------------------------------------------------------------------------------
 * returns the cost of the cheapest path, or -1 if no path exists.
 * start : the cell the car is currently at.
 * end : the cell we would like to drive to.
 * curOrientation : specifies which way the car is currently facing (NORTH, EAST, ...).
 * path : used to "return" the path associated with the least cost, therefore
 * passed by reference. path is a vector that contains the directions, e.g.
 * [STRAIGHT, RIGHT, RIGHT, LEFT]
 * world is the grid that contains 'X' for a building or '0' for a street.
 * markers is a grid of ints that keep track how often we have passed a cell. The
 * coordinates of the markers grid correspond to the world grid.
 */
int findCheapestPath(Cell start, Cell end, Orientation curOrientation,
                     Vector<NextDirection> & path, Grid<char> & world, Grid<int> & markers) {

    // Base Cases:
    // check whether start and end cells are in the world
    if (!world.inBounds(start.row, start.col) || !world.inBounds(end.row, end.col)) return -1;

    // check whether start and end Cells are on street cells
    if (world[start.row][start.col] != street || world[end.row][end.col] != street) return -1;

    // check whether the field has been visited at most once so far
    if (markers[start.row][start.col] > 1) return -1;

    // check whether start equals end. If so, then we have found the shortest path.
    if (start.row == end.row && start.col == end.col) return 0;

    markers[start.row][start.col]++;
    int leastCost = -1;
    Vector<NextDirection> leastCostPath;

    // try all possible actions left, straight, right
    for (NextDirection dir = LEFT; dir <= RIGHT; ++dir) {
        Orientation newOrientation = computeNewOrientation(curOrientation, dir);
        Cell adjacentCell = computeAdjacentCell(start, newOrientation);
        Vector<NextDirection> recursivePath;
        int cost = findCheapestPath(adjacentCell, end, newOrientation, recursivePath,
              world, markers);

        if (cost != -1) { // if path found
             // add the cost of this action depending on the direction
            cost += computeActionCost(dir);
            if (leastCost == -1 || cost < leastCost) {
                leastCost = cost;
                leastCostPath = recursivePath;
                leastCostPath.insert(0, dir);
            }
        }
    }

    markers[start.row][start.col]--;
    path = leastCostPath;
    return leastCost;
}
```