```java
/* This class uses a partially hollow 2D array to represent the games grid.
 * Row and column corresponds to the 2D array row and column respectively.
 * Hence, for the standard grid both row and column must be in the range
 * 0 to 10. Furthermore, either row or column must be 0, 5 or 10.
*/

public class Grid {
    private int multiplier;
    private int roadSize;
    private int mapSize;// column
    private Cell road[];////////////// roads
    private Cell map[][];///////// whole map

    // Set difficulty
    public Grid(int difficulty) {
        this.multiplier = difficulty;
        this.roadSize = (2 + multiplier) * (10 + 9 * multiplier);
        this.mapSize = 6 + (5 * multiplier);
        this.road = new Cell[roadSize];
        this.map = new Cell[mapSize][mapSize];
        int k = 0;
        for (int i = 0; i < mapSize; i++)
            for (int j = 0; j < mapSize; j++)
                if ((i % 5 == 0) || (j % 5 == 0 && i % 5 != 0)) {
                    map[i][j] = new Cell(i, j);
                    road[k++] = map[i][j];
                    if (j % 5 == 0 && i % 5 == 0)
                        map[i][j].gotGold = true;
                } else
                    map[i][j] = null;
    }

    /*
     * Returns a reference to the specified cell. row and cell must be in the range
     * 0 .. 10 and either row or col must be 0, 5 or 10.
```

```
36      */
37     public Cell getCell(int row, int col) throws Exception {
38         if ((row % 5 != 0 && col % 5 != 0) || row < 0 || row >
   10 || col < 0 || col > 10)
39             throw new Exception("Invalid Coordiantes row = " +
   row + " column " + col);
40         return map[row][col];
41     }
42 
43     /*
44      * Returns the cell in the specified direction of the given
   cell. Valid
45      * direction must be either 'R', 'L', 'U', 'D' or ' '. A
   null value will be
46      * returned if attempt to get a non-existent cell.
47      */
48     public Cell getCell(Cell cell, char direction, int presses)
   {
49         if (direction == ' ')
50             return cell;
51         if (direction == 'U') {
52             if (cell.col % 5 == 0 && cell.row > -1 + presses)
53                 return map[cell.row - presses][cell.col];
54             return cell;
55         } else if (direction == 'D') {
56             if (cell.col % 5 == 0 && cell.row < 5 * multiplier +
   6 - presses)
57                 return map[cell.row + presses][cell.col];
58             return cell;
59         } else if (direction == 'L') {
60             if (cell.row % 5 == 0 && cell.col > -1 + presses)
61                 return map[cell.row][cell.col - presses];
62             return cell;
63         } else if (direction == 'R') {
64             if (cell.row % 5 == 0 && cell.col < 5 * multiplier +
   6 - presses)
65                 return map[cell.row][cell.col + presses];
66             return cell;
67         }
68         return null;
69     }
70 
```

```
71      public Cell[] getAllCells() {
72          return road;
73      }
74
75      /*
76       * helper method to check whether val is in the range a to b
77       */
78      private boolean inBetween(int val, int a, int b) {
79          if (val >= a && val <= b)
80              return true;
81          else
82              return false;
83
84      }
85
86      /*
87       * returns the best direction from source cell to the target
    cell. Assumed cells
88       * passed are valid cells in the grid. you need to verify
    this method
89       */
90      public char getBestDirection(Cell from, Cell to) {
91          if (from.row == to.row)
92              if (from.col < to.col)
93                  return 'R';
94              else if (from.col > to.col)
95                  return 'L';
96              else if (from.col == to.col)
97                  if (from.row < to.row)
98                      return 'D';
99                  else if (from.row > to.row)
100                     return 'U';
101
102         int row = to.row;
103         int col = to.col;
104
105         if (inBetween(to.row % 5, 1, 2))
106             row = to.row / 5 * 5;
107         else if (inBetween(to.row % 5, 3, 4))
108             row = to.row / 5 * 5 + 5;
109         if (inBetween(to.col % 5, 1, 2))
110             col = to.col / 5 * 5;
```

```
111         else if (inBetween(to.col % 5, 3, 4))
112             col = to.col / 5 * 5 + 5;
113 
114         if (from.row % 5 == 0)
115             if (from.col < col)
116                 return 'R';
117             else if (from.col > col)
118                 return 'L';
119         if (from.col % 5 == 0)
120             if (from.row < row)
121                 return 'D';
122             else if (from.row > row)
123                 return 'U';
124         return ' ';
125     }
126 
127     /* A helper method to get the absolute value */
128     private int abs(int x) {
129         if (x >= 0)
130             return x;
131         else
132             return -x;
133     }
134 
135     /* A helper method to get the minimum of three values */
136     private int min(int x, int y, int z) {
137         if (x <= y && x <= z)
138             return x;
139         if (y <= z && y <= x)
140             return y;
141         return z;
142     }
143 
144     /*
145      * A method to get the shortest distance from one cell to
    another Assumed cells
146      * are valid cells in the grid
147      */
148     public int distance(Cell from, Cell to) {
149         int d = 0;
150         // compute minimum horizontal distance:
151         if (from.row == to.row)
```

```
152             d += abs(to.col - from.col);
153         else
154             d += min(from.col + to.col, abs(from.col - 5) +
    abs(to.col - 5), abs(from.col - 10) + abs(to.col - 10));
155
156         // compute minimum vertical distance as follows:
157         if (from.col == to.col)
158             d += abs(to.row - from.row);
159         else
160             d += min(from.row + to.row, abs(from.row - 5) +
    abs(to.row - 5), abs(from.row - 10) + abs(to.row - 10));
161         return d;
162     }
163
164     /* Test harness for Grid */
165     public static void main(String args[]) throws Exception {
166         Grid grid = new Grid(1);
167         Cell c1 = grid.getCell(0, 0);
168         Cell c2 = grid.getCell(10, 10);
169         Cell c3 = grid.getCell(0, 2);
170         Cell c4 = grid.getCell(2, 0);
171         Cell c5 = grid.getCell(8, 5);
172
173         System.out.println("Distance from (0,0) to (10,10) is "
    + grid.distance(c1, c2));
174         System.out.println("Distance from (0,0) to (8,5) is " +
    grid.distance(c1, c5));
175         System.out.println("From (0,0) to (0,2) best direction
    is " + grid.getBestDirection(c1, c3));
176         System.out.println("From (0,0) to (2,0) best direction
    is " + grid.getBestDirection(c1, c4));
177     }
178 }
179
```