```
/* 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.
	 */
	public Cell getCell(int row, int col) throws Exception {
		if ((row % 5 != 0 && col % 5 != 0) || row < 0 || row > 10 || col < 0 || col > 10)
			throw new Exception("Invalid Coordiantes row = " + row + " column " + col);
		return map[row][col];
	}

	/*
	 * Returns the cell in the specified direction of the given cell. Valid
	 * direction must be either 'R', 'L', 'U', 'D' or ' '. A null value will be
	 * returned if attempt to get a non-existent cell.
	 */
	public Cell getCell(Cell cell, char direction, int presses) {
		if (direction == ' ')
			return cell;
		if (direction == 'U') {
			if (cell.col % 5 == 0 && cell.row > -1 + presses)
				return map[cell.row - presses][cell.col];
			return cell;
		} else if (direction == 'D') {
			if (cell.col % 5 == 0 && cell.row < 5 * multiplier + 6 - presses)
				return map[cell.row + presses][cell.col];
			return cell;
		} else if (direction == 'L') {
			if (cell.row % 5 == 0 && cell.col > -1 + presses)
				return map[cell.row][cell.col - presses];
			return cell;
		} else if (direction == 'R') {
			if (cell.row % 5 == 0 && cell.col < 5 * multiplier + 6 - presses)
				return map[cell.row][cell.col + presses];
			return cell;
		}
		return null;
	}

	public Cell[] getAllCells() {
		return road;
	}

	/*
	 * helper method to check whether val is in the range a to b
	 */
	private boolean inBetween(int val, int a, int b) {
		if (val >= a && val <= b)
			return true;
		else
			return false;

	}

	/*
	 * returns the best direction from source cell to the target cell. Assumed cells
	 * passed are valid cells in the grid. you need to verify this method
	 */
	public char getBestDirection(Cell from, Cell to) {
		if (from.row == to.row)
			if (from.col < to.col)
```

```
                                return 'R';
                        else if (from.col > to.col)
                                return 'L';
                        else if (from.col == to.col)
                                if (from.row < to.row)
                                        return 'D';
                                else if (from.row > to.row)
                                        return 'U';

                int row = to.row;
                int col = to.col;

                if (inBetween(to.row % 5, 1, 2))
                        row = to.row / 5 * 5;
                else if (inBetween(to.row % 5, 3, 4))
                        row = to.row / 5 * 5 + 5;
                if (inBetween(to.col % 5, 1, 2))
                        col = to.col / 5 * 5;
                else if (inBetween(to.col % 5, 3, 4))
                        col = to.col / 5 * 5 + 5;

                if (from.row % 5 == 0)
                        if (from.col < col)
                                return 'R';
                        else if (from.col > col)
                                return 'L';
                if (from.col % 5 == 0)
                        if (from.row < row)
                                return 'D';
                        else if (from.row > row)
                                return 'U';
                return ' ';
        }

        /* A helper method to get the absolute value */
        private int abs(int x) {
                if (x >= 0)
                        return x;
                else
                        return -x;
        }

        /* A helper method to get the minimum of three values */
        private int min(int x, int y, int z) {
                if (x <= y && x <= z)
                        return x;
                if (y <= z && y <= x)
                        return y;
                return z;
        }

        /*
         * A method to get the shortest distance from one cell to another Assumed cells
         * are valid cells in the grid
         */
        public int distance(Cell from, Cell to) {
                int d = 0;
                // compute minimum horizontal distance:
                if (from.row == to.row)
                        d += abs(to.col - from.col);
                else
                        d += min(from.col + to.col, abs(from.col - 5) + abs(to.col - 5), abs(from.col - 10)
+ abs(to.col - 10));

                // compute minimum vertical distance as follows:
                if (from.col == to.col)
                        d += abs(to.row - from.row);
                else
                        d += min(from.row + to.row, abs(from.row - 5) + abs(to.row - 5), abs(from.row - 10)
+ abs(to.row - 10));
                return d;
        }

        /* Test harness for Grid */
        public static void main(String args[]) throws Exception {
                Grid grid = new Grid(1);
                Cell c1 = grid.getCell(0, 0);
                Cell c2 = grid.getCell(10, 10);
                Cell c3 = grid.getCell(0, 2);
                Cell c4 = grid.getCell(2, 0);
                Cell c5 = grid.getCell(8, 5);

                System.out.println("Distance from (0,0) to (10,10) is " + grid.distance(c1, c2));
                System.out.println("Distance from (0,0) to (8,5) is " + grid.distance(c1, c5));
                System.out.println("From (0,0) to (0,2) best direction is " + grid.getBestDirection(c1,
c3));
                System.out.println("From (0,0) to (2,0) best direction is " + grid.getBestDirection(c1,
c4));
        }
}
```