Double Vision

The DoubleVision company designs inks and fonts that can be easily read by both humans and machines. They design their fonts on a rectangular grid. Shown below is a very simple 5x3 design for the first five digits.

```
.o. .o. oo. oo. o.o
o.o .o. ..o ..o o.o
o.o .o. .o. oo. ooo
o.o .o. o.. ..o ..o
.o. .o. ooo oo. ..o
```

The ink appears to be normal black ink, but just underneath the surface DoubleVision adds a special polymer that can be detected by an infrared scanner. A human sees the black ink but not the polymer, and a machine sees the polymer but not the black ink. The only problem is that the polymer is much more expensive than the ink, so DoubleVision wants to use as little of it as possible. They have discovered that with many fonts, each symbol can be uniquely identified by at most two pixels. By only adding the polymer to one or two pixels per symbol, they drastically lower costs while still ensuring 100% accuracy in their scanners. The font shown above has this property; pixels that uniquely identify each letter are highlighted with '#'. (There are other choices that would work as well.)

```
.#. .o. #o. oo. o.#
#.o .#. ..o ..o o.o
o.o .o. .o. #o. ooo
o.o .o. #.. ..o ..o
.o. .o. ooo #o. ..o
```

Your job is to write a program to determine if a given font has this property, and if so highlight the pixels.

The input consists of one or more test cases, followed by a line containing '0 0 0' (three zeros) that signals the end of the input. Each test case begins with a line containing three positive integers *n*, *r*, and *c*, separated by a space: *n* is the number of symbols in the font, *r* is the number of rows in each grid, and *c* is the number of columns in each grid. The next *r* lines contain the image of each symbol, using the exact format shown in the examples: a dot '.' represents an empty part of the grid, a lowercase 'o' represents a pixel, and adjacent grids are separated by a space. The total width of each line will be at most 79 characters (not counting end-of-line characters), and *r* will be at most 10. The test cases are implicitly numbered starting with 1.

For test case *i*, first output a line that says 'Test *i*'. Then determine if each symbol can be uniquely identified with one or two pixels. If not, output a line with the word 'impossible'. Otherwise, output the font in the same format except that the identifying pixels for each symbol are replaced with '#'.

In general there may be several different pixels or pixel pairs that uniquely identify a symbol. To ensure that the output is unique, we add the following definition and rules. When comparing two pixels, the *topmost-leftmost* pixel is the one closest to the top of the grid. If both pixels are on the same row, then the topmost-leftmost is the one closest to the left of the grid.

If one pixel will work, highlight the topmost-leftmost pixel that works. Never highlight a two-pixel solution if a one-pixel solution is possible. If two pixels are needed, highlight the pair with the

topmost-leftmost pixel. If two or more pairs have the same topmost-leftmost pixel, highlight the one with the topmost-leftmost *other* pixel.

Input:

```
3 2 2
oo oo .o
o. .o o.
3 2 2
oo oo .o
o. .o oo
5 5 3
.o. .o. oo. oo. o.o
o.o .o. ..o ..o o.o
o.o .o. .o. oo. ooo
o.o .o. o.. ..o ..o
.o. .o. ooo oo. ..o
1 2 4
.o..
...o
0 0 0
```

Output:

```
Test 1
impossible
Test 2
#o #o .o
#. .# ##
Test 3
.#. .o. #o. oo. o.#
#.o .#. ..o ..o o.o
o.o .o. .o. #o. ooo
o.o .o. #.. ..o ..o
.o. .o. ooo #o. ..o
Test 4
.#..
...o
```