Tic-Tac-Toe A.I.

Problem Text:

You've been tasked with creating the AI for your company's next blockbuster command line game: Tic-Tac-Toe. Because you've rightly identified command line gaming as the Next Big Thing(tm), you've decided to abstract out the game's AI into its own command line program for easy testing.

In your favorite programming language, write a program that, when given the state of the tic-tac-toe board and the player who is making the move, returns the a list of moves sorted from best to worst move.

The program should read input from standard in and output the results to standard out. Both inputs and outputs should be valid JSON. The input should be a list whose elements are summaries of a given game's state, encapsulated by the following required parameters:

- `board`: The current state of the board, encoded as described below.
- `player`: The player whose turn it is. Can be one of 'x' or 'o'.

A valid JSON dictionary would looke like so:

```
[
  {
    "board": "****x****",
    "player": "o"
  },
```

```
  {
    "board": "o***xo*x*",
    "player": "o"
  },
  {
    "board": "*oo*xx***",
    "player": "x"
  }
]
```

Outputs are also JSON. When all parameters are correctly formed and the program runs without complaint (exit code 0), the output JSON dictionary indexes of the possible moves (a board's index is also described below in the section on board encoding). The order of this list should be sorted descending on how good the move is for the player.

A valid output for the above input would be:

```
[
  {
    "indexes": [
      8,
      6,
      2,
      0,
      7,
      5,
      3,
      1
    ]
  },
  {
    "indexes": [
```

```
      1,
      2,
      6,
      8,
      3
    ]
  },
  {
    "indexes": [
      3,
      0,
      8,
      7,
      6
    ]
  }
]
```

NOTE: You'll probably have some way of scoring any given move, and given the nature of the game, some moves will be equally as good as others, so ties should be broken be doing a secondary sort ordering descending by the move index (so in the first set of indexes above, assuming move 8 and move 6 were scored equally, 8 comes first because it's "higher").

If one of the input keys is missing or otherwise malformed / invalid, the program finishes with exit code 1 and an error message specifying what went wrong (again, in valid JSON).

So on the input:

```
[
  {
```

```
    "board": "**x***o*x"
  }
]
```

The output would be:

```
[
  {
    "message": "invalid player"
  }
]
```

Note that a board in an end state can be submitted, but is not an error. When a board in an end state is submitted, a message for that entry should be supplied. For example, for the input:

```
[
  {
    "player": "o",
    "board": "x*xoxo*ox"
  }
]
```

The output would be:

```
[
  {
    "message": "board at end state."
  }
```

```
]
```

Board Encoding

The board is encoded as a string of 9 characters. Each character of the string represents a space on the board and can be either an 'x', and 'o', or an '*', indicating the space is empty and avaiable for a move. The index of the character of the string maps to a space on the board like so:

```
0 1 2
3 4 5
6 7 8
```

Thus, a board that looks like this:

```
* x *
o x *
* o x
```

Would be encoded as: `*x*ox**ox`

A move is just a reference to the space's index in the string. Given this board as a parameter, the program might return the index of any open space (the asteriks), so here it would be one of 0, 2, 5, or 6.