Overview:

Chapters 10 of the textbook develop and explain a functioning Human verses Computer game of Tic-Tac-Toe. Your task it to extend the game perpendicular to the board.

In 3D Tic-Tac-Toe, a player wins when getting three-in-a-row in the usual way on any one of the three levels. A player can also win by getting three-in-a-row vertically, either in a single column or on a diagonal. Additionally, the middle space on the middle level is banned. For example, each board below shows a win case for player 0:

```
x-- | x-x | ---
--- | 0 x | ---
--- | 000 | ---
```

```
-o- | -o- | -o-    
-o- | - - | -x-   
-x- | ---- | -xx   
```

```
o-- | oox | xx0
--- | -  | ---
--- | --x | ---
```

```
o-o | --x | x--
--- | - o | ---
--- | oxx | -xo
```

From left to right, the levels are numbered 1, 2 and 3. Within a level, the spots are numbered like the keyboard’s number pad.
Grading Rubric [20 points total]:

[File Name: 1 point]: Attach one file in Blackboard with the file name:
CS105_Lab5_yourFirstName.yourLastName.py

[Drawing the Board: 2 points]: The given 3x3 board is way too huge. You must
draw your 3x3x3 after each move in the format given above.

[Getting Input: 2 points]: The given 3x3 game asks the player to specify a move
by inputting a number form 1 through 9. Your game has 3 levels, so the user
not only needs to enter a location 1 through 9, but also must input a level 1
through 3.

Here is a function you may use to help:

```python
#This function inputs multiple integers delimited
# by commas and returns them as a tuple of two integers.
def inputTuple(prompt):
    str = input(prompt)
    str = '(' + str + ')
    values = tuple(int(x) for x in str[1:-1].split(', '))
    return values;
```

Below is an example of how to use the `inputTuple` function (a tuple, is
similar to a list):

```python
playerMove = inputTuple("Enter move as: spot,level")
spot = playerMove[0]
level = playerMove[1]
```

Then, the person running this program might enter 8,3 to specify a move in
spot 8 of level 3.

[No Cheating: 3 points]: Each AI turn, your AI must make a legal move and the
game must always prevent the player from making an illegal move.

[Recognizing a Win: 4 points]: Your game must recognize when either the
player or the AI has attained three-in-a-row, must report the winner and
must query for a new game.

[Taking a Win: 4 points]: Your AI does not need to be super smart; however, if
the AI can win on its current turn, then the AI must make a winning move.
[Blocking a Win: 4 points]: If your AI cannot win in a single move, and the player can win on his or her next move, then the AI must block at least one possible three-in-a-row that the player could make.

Extra Credit:

[Getting Smarter: +5]: Make your AI significantly smarter than the base requirement by adding a key additional strategy.

[Wizard: +5]: After earning the +5 for Getting Smarter, make your AI even smarter by adding yet another key strategy.

[Graphics: +10]: Replace the text printing of the board with a drawn board using Turtle Graphics.

[4×4×4: +20]: Replace the 3×3×3 game requirements with a program that plays 4×4×4 Tac-Tac-Toe AND teach it to play at the Wizard level as defined above for the 3×3×3 game. Note, 4×4×4 Tac-Tac-Toe does not have any banded spots on any levels.
Spoilers:
In the book version, the board is initialized as empty with the python statement:

```python
board = [' '] * 10
```

This makes a list of ten space characters:

```
(index) 0 1 2 3 4 5 6 7 8 9
(list) [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']
```

Where `board[0]` is not used.

For example, the board layout
```
X
X O O
X
```

would have the list representation of:

```
(index) 0 1 2 3 4 5 6 7 8 9
(list) [' ','X',' ','','X','O','O','X',' ',' ']
```

Our 3D Tic-Tac-Toe is just three 2D boards. So where the book uses a list of size 10, we can use a list of size 30. Since you need to print your board with '-' in the empty locations, you would want to set your initial board to all '-'. Thus, create your board with:

```python
board = ['-'] * 30
```

Where the game locations corresponding to the `board` list indices are:

```
  7 8 9 |  17 18 19 |  27 28 29
  4 5 6 |  14 15 16 |  24 25 26
  1 2 3 |  11 12 13 |  21 22 23
```

Then, set the unused elements to the space character:

```python
```

Note: In order for the textbook's `def isBoardFull(board)` to work with minimal changes, it is important to replace each unused element with a character other than '-' (use to show the element corresponds to an open move location).

This representation of the board would mean that when the user enters 7,1 you need to check if `board[7]` is empty. When the user enters 7,2, you need to check if `board[17]` is empty and when the user enters 7,3, you need to check if `board[27]` is empty. Similarly, input 5, 3, must check `board[25]`, and so on.