IT 327
Califf

Program 3
Object-Oriented Programming in Python (with some 2d lists)

The Problem

You are going to write a program that checks chess moves for legality and displays possible moves a chess piece might make (**ignoring** check and more sophisticated moves such as castling and en passant – do not worry about those). If you are not familiar with the details of all chess piece moves or if you think you might benefit from a refresher, check out https://en.wikipedia.org/wiki/Rules_of_chess, focusing on the basic moves.

The Classes

You will start by writing a parent class for all of our different chess pieces called ChessPiece. Then you will write Knight, WhitePawn, Rook, Bishop, and Queen classes. I have provided a King class as an example and a BlackPiece class (which is used as a dummy class for all black pieces). We will only test with white pieces. I have also provided a Board class, a file with some enums, and a program file for you. You can find all of the provided source code on the Linux machines at /home/ad.ilstu.edu/mecalif/public/it327/Program3. You will also find sample input and output files there.

The `ChessPiece` Class

This class must be in chess_piece.py (note the imports in the provided files).

ChessPiece must have exactly the following instance variables (attributes/fields):

- `_row`: an integer specifying the row of the board the piece is on
- `_col`: an integer specifying the column of the board the piece is on
- _label: the enum that represents the piece's label (PieceInfo.EMPTY by default)
- `_color`: the enum that represents the piece's color (BoardInfo.WHITE or BoardInfo.BLACK)

This class must have at least the following methods. You may wish to add additional methods that have logic needed by multiple subclasses (though there are other ways to handle the desired code reuse):

- An __init__ method that takes in values for all four instance variables.
- A `move` method that takes the new row and column as parameters and updates the piece's location instance variables.
- A `get_color` method that returns `_color`.
- A `get_label` method that returns `_label` and takes no parameters.

- An `is_legal_move` method that returns false and takes three parameters: an integer representing the destination row, an integer representing the destination column, and a `Board` object, which is the chess board we're using.
- A `generate_legal_moves` method that takes two parameters: a 2D list of one-letter strings that we will use for showing where the piece can move and a `Board` object. The method will simply return the 2D list of strings.

Subclasses
You will then create the following descendants of ChessPiece. Place these in a file named my_pieces.py. I highly recommend doing them in this order.

- `Knight`
- `Rook`
- `WhitePawn` (because pawns move in only one direction, the different colors behave differently, unlike other pieces)
- `Bishop`
- `Queen`

As you create your subclasses, you should reference my implementation of the `King` class and the test files I am providing as well as the rules for the piece. Do remember that most pieces cannot move through other pieces as you work on your logic. Note also that NONE of the pieces you're creating logically use a nested for loop of the same kind that the King class uses. Don't copy that part.

I have provided an example input file and the corresponding output file for testing the King class for you to use first to ensure that you have ChessPiece working with the King and BlackPiece classes and then as an example for doing your own testing of other pieces. Note that as you add classes, you will need to add appropriate lines to the `make_piece` function in `chess_move_checker`.py as well as adding appropriate import statements at the top of that file. There is a comment in the function where you need to make your changes there. Imports go at the top.

Running your program:
The input and output files are provided as command line arguments, and the python program you are running is chess_move_checker.py, so to run with the sample king input and check the results, you would use

```
python3 chess_move_checker.py king_infile my_outfile
diff king_outfile my_outfile
```

I want efficient implementations, so you are not allowed to call generate_legal_moves from is_legal_move or vice versa. However, you should avoid unnecessary code duplication between

classes, and you might consider whether you can come up with ways to reduce code duplication between generate_legal_moves and is_legal_move that don't introduce inefficiencies.

Submit a zip file containing the .py files that you have written or modified for the assignment. You must not modify `chess_utils.py`, `board.py`, or `mec_pieces.py` and you may only modify the `make_piece` function and the imports in `chess_move_checker.py`.

Grading will be in accord with the program grading criteria available on Canvas.