Chess

In this homework, you should create a chess console game. The initial template of the code is provided and it should be extended as described below.

The Game

The game is divided into 3 files and several classes. Each class is described below. You can make minor changes (Ex. add arguments to functions, define extra methods, add attributes to classes and such), but the general structure should remain unchanged (Removing, not using, or changing given methods too much is not allowed).

Pieces

At first, we need to define chess pieces - `pieces.py` contains several classes for that. The first class `Piece` is a general class, that contains basic functionality. `Piece` class should have the following attributes and properties:

- **color** - color should be property (not attribute). It can have only two values: "White" or "Black" (strings). It should **not** have a setter, as color can not be changed after Piece is created.
- **position** - position is a tuple of character and integer (example ('H',1), ('B',3)). the position should also be a property. It should have a setter function (`@position.setter`), which should check that position is always assigned a valid tuple.
- **board** - the board should be a hidden attribute (not property). It is an instance of Board class, which is described below. Piece needs to know information about the board to check the path before making the move.

`Piece` should also have the following methods:

- **constructor** - Constructor should get color, initial position, and board as arguments and initialize them.
- **checkMove** - This method takes destination as an argument (dest is tuple like position) and it should check if moving from position to destination is possible. First of all, we check if the piece can move to that position (For example, a bishop can only move on diagonals), after that we should check that path to the destination is not blocked. The path can be blocked by figures of the same or different color. Pieces of different colors can be killed and you should take care of that too. For example, if you want to move the Rook 1 cell forward, but there's another player's piece then you can kill it, but you can not jump over it. `checkMove` should return True or False.
- **move** - This method also takes destination as an argument. You should use the `checkMove` function to check that move is possible and if it is you should update both - position and gameBoard from Board. `move` should also return True if the move was made or False otherwise.
- **getName** - This method returns the name of the class in string form.
- **getIcon** - This method returns an appropriate icon, which can be used later to display the board in the console. Icons can be found in `whiteIcons` and `blackIcons` dictionaries. Each dictionary contains the class name, icon pairs. For example, if you want to get an icon for white `Pawn`, you can get it with `whiteIcons["Pawn"]`.

As Piece is not really a chess piece it can not make move so `move` and `checkMove` functions should always return false, but they can be implemented later for other types. `getIcon` can return `None` for the same reason.

After that, you should implement actual pieces: `Knight`, `Rook`, `Bishop`, `Queen`, `King`, `Pawn`. Each piece has its class which is the child class of `Piece`. Hence, it inherits all attributes and methods mentioned above. In each class, you need to redefine `getName`, `move`, and `checkMove` methods. You can implement each class without inheritance, but points will be deducted.

You can find the description of movements here: https://www.ichess.net/blog/chess-pieces-moves/

You should implement basic movement and capture for each class, but you do not need to implement special moves: `En Passant` or `Castle`. `Promotion` should be allowed, but it will be part of the `Chess` class.

Board

`Board` class stores all the pieces. You can use the dictionary, list of lists, or any other structure for storing pieces, it's up to you.

`Board` should have the following methods:

- **constructor** - Constructor should initialize board and it should call `placePieces` method.
- **placePieces** - This method should populate the board with pieces. (You can find the initial setup here: https://www.chess.com/article/view/how-to-set-up-a-chessboard).
- **setPiece** - This method gets position and a new piece as arguments and should update the board accordingly.
- **getPiece** - This method gets the only position as an argument and should return the appropriate piece or `None` if it does not exist.
- **makeMove** - This method gets 3 arguments: player, start position, and end position. You should check if the player can make that move and update the board according to it if it is possible. At first, make sure that the piece exists on the start position and it belongs to the player (Player is also string "White" or "Black"). Then you can check if the move is valid using functions from Piece classes.
- **displayBoard** - This method should print the board in the console. You can use icons from Piece classes and your own ASCII art or just follow the example.

Chess

`Chess` is the main class, it should contain two attributes **board** and **currentPlayer**. 'board' is an instance of `Board` class, used to store game state and 'currentPlayer' is a string: "White" or "Black". The `Chess` class should have the following methods:

- **constructor** - Constructor should initialise 'board' object and set 'currentPlayer' to "White".
- **swapPlayers** - This method should change 'currentPlayer': "Black" to "White" and "White" to "Black".
- **isStringValidMove** - This method takes a string as an argument and checks if it describes a move or not. The valid string should contain start position, space, and end position. Each position should be represented with a letter and a number. For example, valid strings are: "H3 A8", "B2 C2", "E5 C7", "D1 D8", "A1 A1". Invalid strings are: "K2 A1", "A7 A9", "A1A2", "G3 H", "G2-G3", "A2 A3 ", "bla".
- **play** - This method is the start point of the game. Whenever it is called, the infinite loop starts. At each iteration following things should happen:
  1. Display board in the terminal using the `board.displayBoard` method.
  2. Print whose turn it is and ask for the input.
  3. Read input from the terminal
  4. If the input is "EXIT" stop the loop and finish the game.
  5. If not, check if the input is valid input using `isStringValidMove` method and if it is not go back to step 2.
  6. If input describes a valid move, extract start and end positions from the string and try to make move using `board.makeMove` method. If the move was not successful, go back to step 2.
  7. After the move was made, check if Pawn was promoted.
     1. If Pawn was promoted ask the user, to which piece he/she want to change it (For example, if the user enters "Rook" change Pawn with Rook). You can find more information about the promotion here: https://www.chess.com/terms/pawn-promotion.
     2. If the input does not match with any possible piece type ask it again until you get valid input.
  8. At the end, swap the current player using `swapPlayers` method and continue the game.

Examples

Here are some examples of what your game should look like:

The board:

```
   (1)(2)(3)(4)(5)(6)(7)(8)
(A)[♜][♞][♝][♛][♚][♝][♞][♜]
(B)[♟][♟][♟][♟][♟][♟][♟][♟]
(C)[ ][ ][ ][ ][ ][ ][ ][ ]
(D)[ ][ ][ ][ ][ ][ ][ ][ ]
(E)[ ][ ][ ][ ][ ][ ][ ][ ]
(F)[ ][ ][ ][ ][ ][ ][ ][ ]
(G)[♙][♙][♙][♙][♙][♙][♙][♙]
(H)[♖][♘][♗][♕][♔][♗][♘][♖]
```

The board after making the move:

```
White's turn. Enter a move:
B2 C2
   (1)(2)(3)(4)(5)(6)(7)(8)
(A)[♜][♞][♝][♛][♚][♝][♞][♜]
(B)[♟][ ][♟][♟][♟][♟][♟][♟]
(C)[ ][♟][ ][ ][ ][ ][ ][ ]
(D)[ ][ ][ ][ ][ ][ ][ ][ ]
(E)[ ][ ][ ][ ][ ][ ][ ][ ]
(F)[ ][ ][ ][ ][ ][ ][ ][ ]
(G)[♙][♙][♙][♙][♙][♙][♙][♙]
(H)[♖][♘][♗][♕][♔][♗][♘][♖]
```

Invalid move:

```
   (1)(2)(3)(4)(5)(6)(7)(8)
(A)[♜][♞][♝][♛][♚][♝][♞][♜]
(B)[♟][ ][♟][♟][♟][♟][♟][♟]
(C)[ ][♟][ ][ ][ ][ ][ ][ ]
(D)[ ][ ][ ][ ][ ][ ][ ][ ]
(E)[ ][ ][ ][ ][ ][ ][ ][ ]
(F)[ ][ ][ ][ ][ ][ ][ ][ ]
(G)[♙][♙][♙][♙][♙][♙][♙][♙]
(H)[♖][♘][♗][♕][♔][♗][♘][♖]
Black's turn. Enter a move:
A3 C1
INVALID move. Piece should be black!
```

Invalid move:

```
   (1)(2)(3)(4)(5)(6)(7)(8)
(A)[♜][♞][♝][♛][♚][♝][♞][♜]
(B)[♟][ ][♟][♟][♟][♟][♟][♟]
(C)[ ][♟][ ][ ][ ][ ][ ][ ]
(D)[ ][ ][ ][ ][ ][ ][ ][ ]
(E)[ ][ ][ ][ ][ ][ ][ ][ ]
(F)[ ][ ][ ][ ][ ][ ][ ][ ]
(G)[♙][♙][♙][♙][♙][♙][♙][♙]
(H)[♖][♘][♗][♕][♔][♗][♘][♖]
Black's turn. Enter a move:
H1 E1
INVALID move. Path is blocked!
```

Grading

This assignment is worth 10 points in total.

- Pieces - 4 points in total
  - Piece class - 1 point if everything is correct, 0.5 points if you have minor mistakes or do not use properties.
  - Knight, Rook, Bishop, Queen, King, Pawn - 2 points if everything is correct, 1 points if you have minor mistakes.
  - Inheritance - 1 point if Piece is parent class of other classes and Inheritance is used properly.
- Board - 3 points in total
  - Constructor and placePieces - 1 points
  - setPiece, getPiece and attribute for storing pieces - 1 points
  - makeMove - 0.5 points
  - displayBoard - 0.5 points
- Chess - 3 points in total
  - Constructor, swapPlayers - 0.5 points
  - isStringValidMove - 0.5 points
  - play - 2 points

If the code does not compile and run, your code **will be graded with 0 points**.