Position 1:

Board Evaluation: 0

Piece Under Attack = 3 | Bishop

Defenders = {3, 9} | {Knight, Queen}

Attackers = {3, 3} | {Knight, Bishop}

Board Evaluation: -3

Piece Under Attack = 3 | Knight

Defenders = {3} | {Bishop}

Attackers = {3, 9} | {Knight, Queen}

Board Evaluation: 0

Piece Under Attack = 3 | Knight

Defenders = {9} | { Queen}

Attackers = {3} | {Bishop}

Board Evaluation: -3

Piece Under Attack = 3 | Bishop

Defenders = {} | None

Attackers = {9} | {Queen}

Board Evaluation: 0

Stable

If there are the same number of defenders as attackers, the last defender will remain unharmed, the trades are then the Piece under attack with Attackers[0], then Defenders[0] with Attackers[1], with the pattern continuing as Defenders[i] is compared with Attackers[$i + 1$], until $i + 1 =$ the last index of the Attackers array. Let us see two examples where in one, one of these comparisons is in favor of Attackers, and in another, one of these comparisons is in favor of Defenders.

The left is attackers favor, this is not stable. The right is defenders favor and is in fact stable, as the only trade possible is an even one. We now know the difference between different comparisons of this style.

Let us now look at positions in which the number of defenders and attackers are not the same.

Position 2:

Board Evaluation: 0

Piece Under Attack = 3 | Bishop

Defenders = {1, 3, 3, 9} | {Pawn, Bishop, Knight, Queen}

Attackers = {3, 3} | {Bishop, Knight}

Board Evaluation: -3

Piece Under Attack = 3 | Bishop

Defenders = {3} | {Knight}

Attackers = {1, 3, 9} | {Pawn, Knight, Queen}

Board Evaluation: 0

Stable

In this example, the Pawn under attack is stable because it has more defenders than attackers.

Piece Under Attack = 1 | Pawn

Defenders = {3, 9} | {Knight, Queen}

Attackers = {3} | {Knight}

In this other example, the Pawn under attack is stable and it has the same number of defenders as attackers.

Defenders = {3, 9} | {Knight, Queen} | Attackers = {3, 3} | {Knight, Knight}

The next position shows an example of a Pawn under attack by pieces of greater value than it that is unstable, as there are more attackers than defenders.

Position 3:

Board Evaluation: 0

Piece Under Attack = 1 | Pawn

Defenders = {3, 9} | {Knight, Queen}

Attackers = {3, 3, 9} | {Knight, Knight, Queen}

Board Evaluation: -1

Piece Under Attack = 3 | Knight

Defenders = {3, 9} | {Knight, Queen}

Attackers = {3, 9} | {Knight, Queen}

Board Evaluation: 2

Piece Under Attack = 3 | Knight

Defenders = {9} | {Queen}

Attackers = {3, 9} | {Knight, Queen}

Board Evaluation: -1

Piece Under Attack = 3 | Knight

Defenders = {9} | {Queen}

Attackers = {9} | {Queen}

Board Evaluation: 2

Piece Under Attack = 9 | Queen

Defenders = {} | None

Attackers = {9} | {Queen}

Board Evaluation: -7

Piece Under Attack = None

Not Stable

We can see that the decider of stability has more to do with the number of defenders and attackers than piece value in these cases. Here is another example of the starting position that will also be stable.

In this case, there are more defenders than attackers, and it is stable as the trade would be Pawn for Pawn, then Knight for Knight. In this case of stability, the trade is optional, unlike the other two previous stable cases in which the trade is losing for the attacker. Either way, all we want to find is stability.

Summary:

Let n be the number of attackers, and m be the number of defenders. We will define different cases here using the above examples.

Case 1: $n = m$

If n and m are identical then compare each piece that is trading. First PUA (Piece under attack) with Attackers[0], then Defenders[0] with Attackers[1]… Defenders[i] with Attackers[$i + 1$] where $i + 1 =$ Attackers last index. If any of these comparisons are in favor of the attacker: unstable. Otherwise: stable.

Case 2: $n > m$

If there are more attackers than defenders, the position can be shown to be unstable.

Case 3: $m > n$

If there are more defenders than attackers, the position can be shown to be stable.

Find any positions that break these rules that do not utilize checks or pieces of lower values initially attacking the piece under attack (both of these cases are already known to be unstable).

If you find one, send it and we can analyze and revise.

Some useful functions if you want to attempt to code:

ChessBoard.piece_map()

This gets the map of every piece on the board by square number.

ChessBoard.attackers()

This gets the number of attackers of a certain square and color (white attackers on a white piece could be called defenders, and same for black attackers on a black piece).

https://python-chess.readthedocs.io/en/latest/core.html#

Here is the documentation for other functions you can use, any coding will definitely use the above two functions as well as possibly others, feel free to attempt to code and ask questions, there are people who know quite a bit of the python chess library by now!

Happy hacking!