**calculate\_center\_control**



Center\_control= control\_score + threat\_score

1. The deeper our own pieces advance into the center, the higher the control\_score (with weighting).

2. If an enemy piece occupies a central square and that square is also within our control, the threat\_score increases (since we can attack the enemy’s central piece).

3. If a square does not contain an enemy piece but is under our control (through attack paths), the control\_score also increases.

**piece\_activity\_score (piece activity / mobility)**

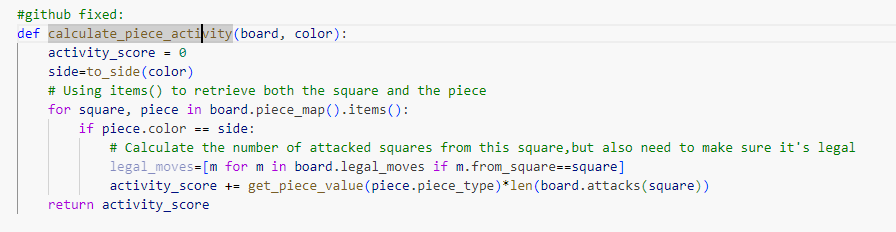
What it measures

The number of squares controlled (mobility) by all of our pieces. The more squares a piece can attack/control, the more active it is.

How the code calculates it

Iterate over every piece of our side on the board, compute len(board.attacks(square)), and sum these counts.

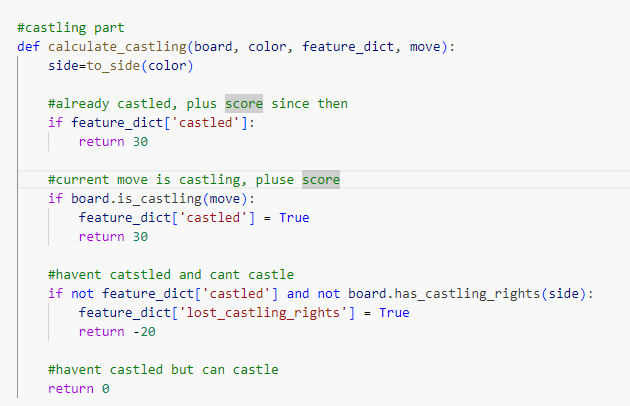
Do this at every move and keep accumulating.



**calculate\_king\_safety**

****

**calculate\_castling**



Castling completed → stable positive score; r**eward the continuation of the castled state (subsequent moves keep +30).**

Castling failed → penalty. -20

Castling opportunity unused → neutral. 0

**calculate\_king\_tropism**

King tropism how much the king was threatened

The score is negative

**Locate our king**

Determine the color and position of our king.

**Iterate over all pieces on the board**

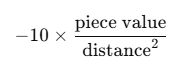
**Skip our own pieces**

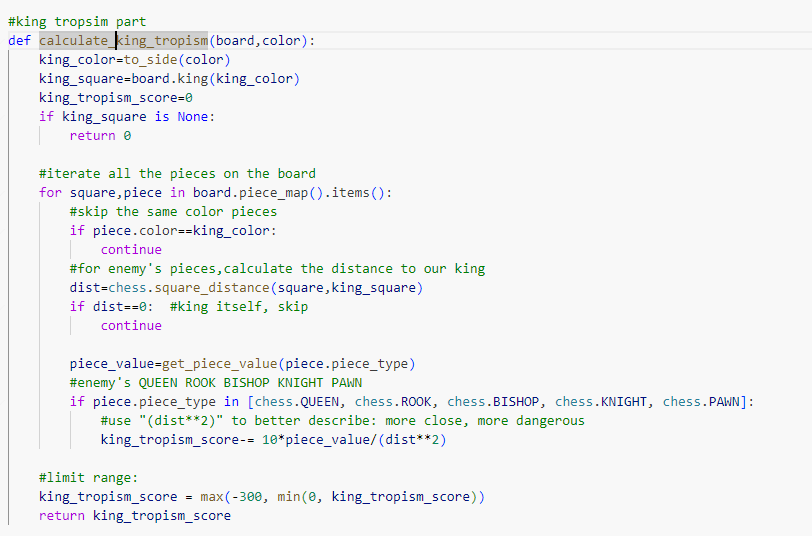
Only consider enemy pieces

**Calculate distance between enemy pieces and the king**

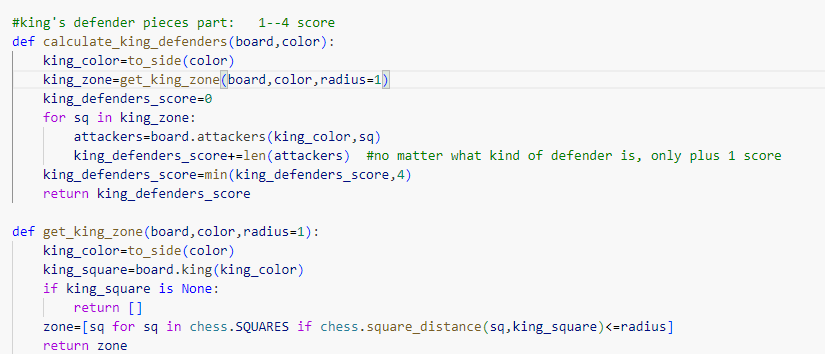
using Chebyshev distance

**Calculate threat value**

King\_tropism\_score **-=** 



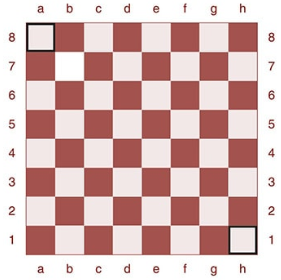
**calculate\_king\_defenders & get\_king\_zone**



**get\_king\_zone** is used to obtain all squares within the specified radius around the king, serving as the area for the defense calculation.

Calculates the number of our own pieces in the **king\_zone**

**Add the score for our pieces defending the king within the king zone, with an upper limit on the total.**

**calculate\_pawn\_shield & is\_open\_file**

1.Pawn Shield Rules:

Rules:

* The pawn shield is evaluated only when the king is still on its back two ranks (ranks 0–1 for White, i.e., the 1st–2nd ranks; ranks 6–7 for Black, i.e., the 7th–8th ranks).
* If the king has already moved forward into the middlegame or endgame, the score is set to 0 and the pawn shield is not considered.

2.Pawn shield definition:

Take the king’s current file and the files immediately to the left and right, for a total of 3 files.

If the king is on the a-file (file = 0), there is no file to the left → only take a and b.

If the king is on the h-file (file = 7), there is no file to the right → only take g and h.

On these 3 files, check the squares directly in front of the king:

White: rank + 1

Black: rank - 1

3.Definition of open file:

A file is considered an open file if there are no friendly pawns on it.

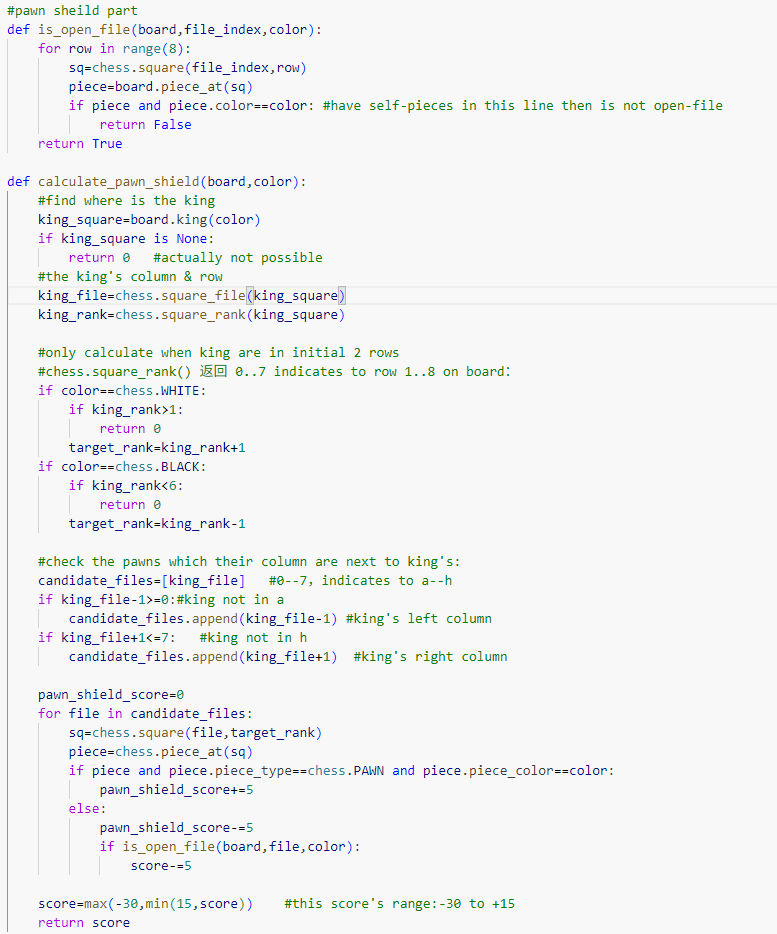
How to determine: check all 8 squares along the file; if any friendly pawn is present, it is not an open file; otherwise, it is an open file.

4. Scoring / Penalty Rules:

Each pawn shield square with a friendly pawn → +5

Each pawn shield square without a friendly pawn → -5

If a pawn shield square is missing a pawn and the corresponding file is an open file → additional -5



**calculate\_zone\_control**

**King Control Zone definition [this is defined on my own]**: Use get\_king\_zone function and additionally include 2–3 ranks in front of the king.

The *get\_king\_zone* function returns the squares immediately surrounding the king, defined as the ***king\_zone***.

The **King\_Control\_Zone,** however, not only i**ncludes the *king\_zone* but also the squares 2–3 ranks in front of the king**

Use .attacks() to determine which pieces control these squares(the King Control Zone) and compare the numbers on both sides

If enemy pieces attack this area, add attack points (weighted by piece type).

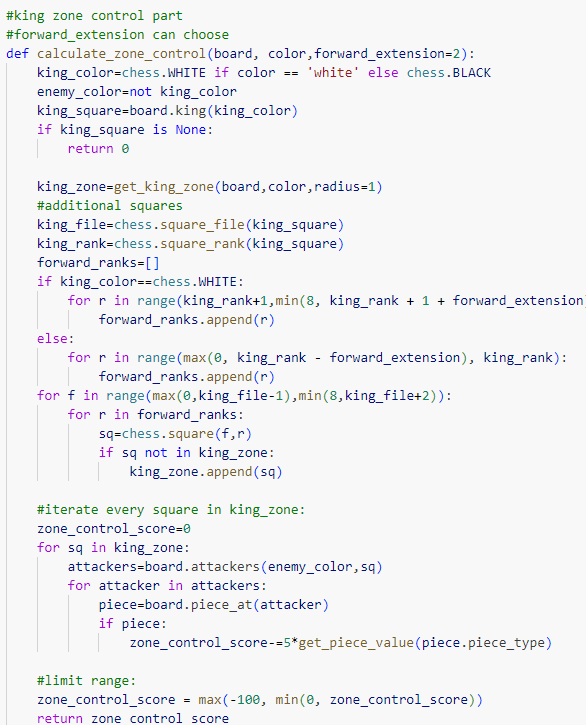
If the number of attacking pieces reaches a certain threshold, the penalty can be amplified

**For each square within the King control zone:**

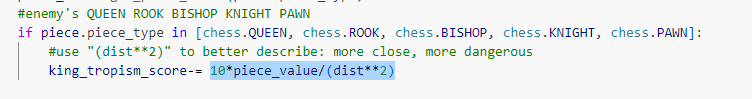
Iterate over all enemy pieces attacking that square.

Add points based on the piece type (negative values indicate a threat to the king).

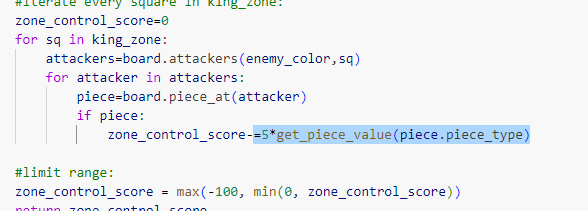
Optionally, apply a slight decay to the score for squares farther from the king.



**COMPARE OF KING TROPISM & ZONE CONTROL**

king\_tropism\_score measures the direct threat of enemy pieces targeting the king itself. This is usually the most dangerous situation, so the penalty is higher **10** (similar to Stockfish, where a queen/rook/bishop/knight attacking the king at close range quickly impacts king safety). ****

calculate\_zone\_control measures the threat of enemy control over the area surrounding the king. This represents a potential threat or spatial exposure; although dangerous, it does not involve a direct attack on the king, so the penalty is relatively milder. **5**

****

**calculate\_diagonal\_exposure**

The idea to calculate this metric was inspired by the logic used in writing the pawn\_shield feature

The pawn\_shield considers whether there are friendly pawns on the files directly in front of the king and on the adjacent files.

However, the king’s safety does not depend solely on the pawn wall in front; it is also closely related to whether the diagonals are exposed:

* If there are no friendly pawns on the king’s diagonals, the king can be easily pressured by enemy bishops or queens from a distance.
* In particular, after short or long castling, if pawns on f2/f7, g2/g7, h2/h7 or c2/c7, b2/b7, a2/a7 are missing, it creates the classic “diagonal weakness.

**calculate\_diagonal\_exposure**

1. King’s position (file, rank).

 If there are no friendly pawns blocking the king’s rank (horizontal) → the king is vulnerable to rooks or queens attacking horizontally → penalty.  
If the two diagonals through the king have no friendly pawns → the king is vulnerable to bishops or queens attacking diagonally → penalty.

2. For each direction:

* If a friendly piece is encountered → consider it safe and stop.
* If an enemy long-range piece (rook, bishop, queen) is encountered with no blocking pieces in between → apply a large penalty based on distance.

 If the entire line has no friendly pawns → apply a small penalty.

 Set upper and lower limits for the score.



**King escape score**

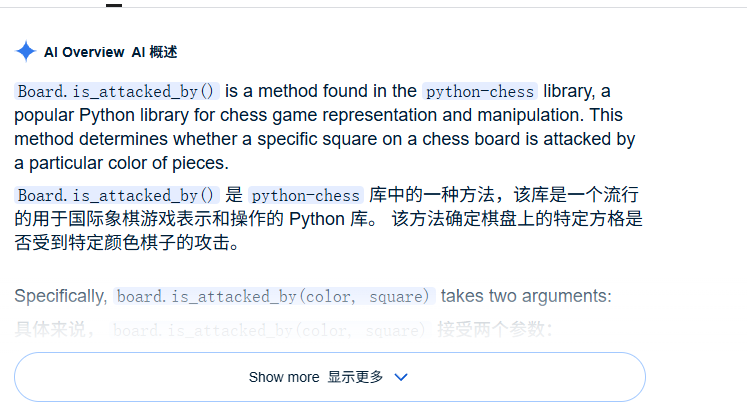
Locate the king’s position.

Use get\_king\_zone to get the squares surrounding the king (may be fewer than 8 at edges and corners).

For each square:

* + Check if it is occupied by a friendly piece → if so, the king cannot move there.
  + Check if the square is under enemy attack → if so, the king cannot move there.
  + Check if the square is within the board → get\_king\_zone already ensures this.

Finally, return the number of available escape squares, multiplied by 5, to contribute to the overall king safety evaluation.



* 1. Each sub-function of king\_safety\_score can also be extracted as a separate feature column.
  2. However, currently, the **scoring units** in each sub-function **need to be standardized**—for example, some functions apply penalties or rewards in units of 10, while others use units of 1.
  3. Additionally, a threshold is often applied when calculating a particular component, such as capping the score contribution at 200, but the exact value of this **threshold still needs to be** agreed upon and **confirmed**