### **Module 1: position.py**

This module will handle position representation, conversion between formats, and basic move generation. It will include the following classes and functions:

* **class Position**: This class will represent a chess position internally.
* **Attributes:**board: A 2D array representing the chessboard and piece placement.
* turn: A variable to store whose turn it is to move (White or Black).
* Other attributes as needed (castling rights, en-passant square, etc.).
* **Methods:**\_\_init\_\_(self, position\_str=''): Initializes a Position object. Optionally takes a string in POSITIONSTR, FEN, or EPD format to set up the position.
* from\_fen\_epd(self, fen\_epd\_str): Sets up the position from a string in FEN or EPD format.
* from\_positionstr(self, position\_str): Sets up the position from a string in POSITIONSTR format.
* to\_epd(self, operation\_list=None): Converts the position to EPD format. Optionally adds EPD operations to the output string.
* mirror(self): Mirrors the position.
* get\_legal\_moves(self): Returns a list of legal moves in the current position.
* **Functions:**
* fen\_epd\_to\_str(fen\_epd\_str): Converts a position in FEN or EPD format to POSITIONSTR format.

### **Module 2: move.py**

This module focuses on move representation and execution. It will contain:

* **class Move**:
* **Attributes:**from\_square: The starting square of the move.
* to\_square: The ending square of the move.
* move\_type: The type of the move (e.g., normal, capture, en-passant, castling).
* **Methods:**\_\_init\_\_(self, from\_square, to\_square, move\_type): Initializes a Move object.
* **Functions:**
* do\_move(position, move): Executes the given move on the provided position.
* do\_move\_ok(position, from\_square, to\_square): Checks if a move is legal and generates black/white moves. Returns True if the move is legal, False otherwise.
* do\_move\_c(position, from\_square, to\_square): Performs a move without requiring move type information. Faster than do\_move\_ok and used when the move is already validated.
* get\_next(position): Finds the next legal move in the position. Updates the fra and til parameters to the next move.

### **Module 3: search.py**

This module will handle the search for the best move in a given position. It will include:

* **class Search**:
* **Attributes:**position: The current position being analyzed.
* depth: The search depth.
* Other attributes as needed (search parameters, transposition table, etc.).
* **Methods:**\_\_init\_\_(self, position, depth): Initializes a Search object.
* find\_best\_move(self): Performs the search and returns the best move found.
* **Functions:**
* find\_trk(position, depth, extra): Finds a move at a specific depth in the search tree. Updates the Traek parameter with the move data.
* get\_move(position, move\_nr, quick=False): Makes a move based on a move number. If quick is True, it drops the check for checks.
* get\_move\_nr(position, from\_square, to\_square, quick=False): Gets a move number based on a move. If quick is True, it drops the check for checks.

### **Module 4: engine.py**

This module will contain the main engine logic and initialization. It will include:

* **Functions:**initialize(): Initializes the chess engine, likely setting up initial variables and data structures.

These Python modules provide a basic framework for a chess engine based on the PL\_PIG\_CHESS\_ENGINE package. They offer modularity and separation of concerns, allowing for easier development and maintenance. [1-6] You can extend these modules by adding more sophisticated search algorithms, evaluation functions, and user interface elements. Please note that this is a simplified representation based on the given information. Further details might be needed for a full implementation.