{ board: piece array array ; transposition_table: (hashcode*, (move_sequence* int for score)) hashtable ; history_table: same as transposition_table ; alive_pieces: (piece, position) hash table ; previous_moves: move_sequence ; predicted_moves: move_sequence ; undo: boolean }

(* Type needed: ① Position: (x, y).

② piece: ("_", color).
↑ one character

③ hashcode: we need to write our own hash function to hash a state into a hashcode

④.1 move_sequence: a list of ④.2 move. of type ((x1, y1), (x2, y2), piece_captured (↑ option *)).

⑤ overall_record.

Overall input & output:

(* a record of board, transposition table (which contains states that occured), and history table (extremely good / bad states from playing history, or by hard-coding), alive pieces, previous moves (up to N moves, for undo & panelty), predicted moves, undo (whether the player has undo once or not)*).

not necessary feature)

Move Generation: produce a list of (all) possible & legal movements.

① for each piece on the moving side, call their respective movement generation helper function.

Helper functions: one for each of the following:
<1> Pawn: red <2> Pawn: black <3> canon <4> king <5> Advisor
<6> Elephant <7> Rook <8> Horse

② In each helper function above,
first, generate possible moves by the rule one-by-one
second, check if the move is legal [helper function] by checking:
<1> if there is a piece of its own side at the destination
<2> two kings are not directly facing each other
<3> not repeating certain sequence of moves ≥ 3 times. (so need to store 3×2×2=12 previous moves; how about 数照一杀、连续判负?)

* Before move generation, first check if player's move is the same as predicted (if there is one)
if yes, just follow the prediction if there is one.

Search: return the best move sequence with K steps, where K is the desired depth.

☆ Iterative Deepening AlphaBeta Algorithm.

<1> After 1st round of move generation, search through all of them and look into the history table & transposition table. If exists, done; if not, evaluate their result, order by result from high to low.

<2> Starting from the higher order ones, generate (n+1)th steps for a nth step, evaluate and save the benchmark (the worst case for ……).

<3> Go to the next highest nth step, generate its (n+1) th steps, evaluate one-by-one. If one of the (n+1)th step is evaluated to be lower than the benchmark, no need to search the rest of moves for this nth step. Repeat <3> until the last nth steps is evaluated. save the lowest score of all the (n+1)th steps that are evaluated as the score of this nth step. If it is higher than the benchmark, update benchmark to this value. Record down this nth → (n+1)th min sequence.

<4> Order the nth → (n+1)th step results from <3> by their scores from high to low.

Repeat <2> ~ <4>. until nth → (n+1)th → … → (n+k) th is generated, and k = desired depth.

Evaluation: return a score for a state.

Possible helper functions:

① Material Balance
② Mobility & Board Control
③ Development
④ King Safety

Movement: → move a piece, update the board and pieces info, previous_moves, etc.

→ check if it ends game yet.

* → undo a move. (eg. can undo only once, then need an undo flag in overall-record).