```
!pip install python-chess
```

```
Requirement already satisfied: python-chess in /usr/local/lib/python3.10/dist-packages (1.999)
Requirement already satisfied: chess<2,>=1 in /usr/local/lib/python3.10/dist-packages (from python-chess) (1.10.0)
```

```
import chess
import math

class ChessSimulator:
    def __init__(self):
        # Initialize the chess board
        self._board = chess.Board()

    def print_board(self):
        # Print the current state of the chess board
        print(self._board)

    def make_move(self, move_str):
        # Make a move on the chess board
        legal_moves = [str(move) for move in self._board.legal_moves]

        # Check if the move is legal
        if move_str not in legal_moves:
            print("Invalid move. Please try again.")
            print("Legal moves:", legal_moves)
            return False

        try:
            self._board.push_san(move_str)
            return True
        except ValueError:
            print("Invalid move. Please try again.")
            return False

    def evaluate_board(self):
        score = 0
        # Evaluate the board based on the pieces on it
        for square in self._board.piece_map():
            piece = self._board.piece_at(square)
            if piece:
                if piece.symbol().isupper():
                    score += 1
                elif piece.symbol().islower():
                    score -= 1
        return score

    def alpha_beta_pruning(self, depth, alpha, beta, maximizing_player):
        if depth == 0 or self._board.is_game_over():
            return self.evaluate_board()

        # Perform alpha-beta pruning
        if maximizing_player:
            for move in self._board.legal_moves:
                self._board.push(move)
                alpha = max(alpha, self.alpha_beta_pruning(depth - 1, alpha, beta, False))
                self._board.pop()
                if beta <= alpha:
                    break
            return alpha
        else:
            for move in self._board.legal_moves:
                self._board.push(move)
                beta = min(beta, self.alpha_beta_pruning(depth - 1, alpha, beta, True))
                self._board.pop()
                if beta <= alpha:
                    break
            return beta

    def find_best_move(self, depth):
        best_move = None
        max_eval = -math.inf
        alpha = -math.inf
        beta = math.inf
        for move in self._board.legal_moves:
            self._board.push(move)
            eval = self.alpha_beta_pruning(depth - 1, alpha, beta, False)
            self._board.pop()
            if eval > max_eval:
                max_eval = eval
                best_move = move
        return best_move
```

```
    def play(self, depth):
      # Play the chess game until it's over
      move_counter = 0
      while not self._board.is_game_over():
          if self._board.turn:  # Maximizing player's turn (capital letters)
              move = self.find_best_move(depth)
              self._board.push(move)
              print("Move", move_counter + 1, "- Capital's move:", move)
          else:  # Minimizing player's turn (small letters) - User's turn
              self.print_board()  # Print the board before the user's move
              user_move = input("Your move (small letters): ")
              if not self.make_move(user_move):
                  continue  # Ask the user for input again if the move is invalid
              print("Move", move_counter + 1, "- Your move:", user_move)
          move_counter += 1
          self.print_board()  # Print the board after each move

      print("Game over. Result: " + self._board.result())


if __name__ == "__main__":
    simulator = ChessSimulator()
    depth = 3  # Depth of the search tree
    simulator.play(depth)
```

```
Your move (small letters): f7g8
Invalid move. Please try again.
Legal moves: ['g7h8', 'g7h7', 'g7f7']
r n b q . b r .
p p p . p . k .
. . . . . . . .
. . . p . . Q .
. . . . . . . .
. . . . P . . .
P P P P . P P P
R N B . K B . R
Your move (small letters): g7h8
Move 16 - Your move: g7h8
r n b q . b r k
p p p . p . . .
. . . . . . . .
. . . p . . Q .
. . . . . . . .
. . . . P . . .
P P P P . P P P
R N B . K B . R
Move 17 - Capital's move: g5h5
r n b q . b r k
p p p . p . . .
. . . . . . . .
. . . p . . . Q
. . . . . . . .
. . . . P . . .
P P P P . P P P
R N B . K B . R
r n b q . b r k
p p p . p . . .
. . . . . . . .
. . . p . . . Q
. . . . . . . .
. . . . P . . .
P P P P . P P P
R N B . K B . R
Your move (small letters): f8h6
Move 18 - Your move: f8h6
r n b q . . r k
p p p . p . . .
. . . . . . . b
. . . p . . . Q
. . . . . . . .
. . . . P . . .
P P P P . P P P
R N B . K B . R
Move 19 - Capital's move: h5h6
r n b q . . r k
p p p . p . . .
. . . . . . . Q
. . . p . . . .
. . . . . . . .
. . . . P . . .
P P P P . P P P
R N B . K B . R
Game over. Result: 1-0
```