```
# Main class that describes a poker state. The state includes
# the current hand, pot, community cards, opponent's bet, and whether or
not it's our turn.
class PokerState():
    def __init__(self, playerHand,communityCards,pot,oppBet,playerTurn):
        # Represents our current hand
        self.playerHand = playerHand
        # Represents the community cards, (flop,turn,river)
        self.communityCards = communityCards
        # Represents the total pot
        self.pot = pot
        # Represents the opponent's current bet amount
        self.oppBet = oppBet

        # Boolean that determines if it's our turn or not
        self.playerTurn = playerTurn

# Returns all possible actions a player can take depending on the turn
def generate_possible_actions(state):
    if state.playerTurn:
        return ['bet', 'call', 'fold', 'raise']
    else:
        return ['check', 'call', 'fold']

# Evaluation function that deteremines the hand strength of the player.
# We use functions evaluate_hand to accomplish this.
def evaluate_state(state):
    # Evaluation function based on hand strength and potential winnings
    player_hand_strength = evaluate_hand(state.playerHand,
state.communityCards)
    return player_hand_strength + state.pot

# Returns the evaluation of the current hand given the player's
# current cards and the community cards
# We use the function rank_hand to determine how valuable the hand is
def evaluate_hand(playerHand, communityCards):
    # Combine player hand and community cards
    full_hand = playerHand + communityCards
    hand_strength = rank_hand(full_hand)
    return hand_strength
```

```python
def rank_hand(hand):
    # Simplified ranking: only checks for pairs, three of a kind, and high
card
    values = [card[0] for card in hand]
    value_counts = {value: values.count(value) for value in values}

    # Check for pairs, three of a kind, etc.
    if 4 in value_counts.values():
        return 7  # Four of a kind
    elif 3 in value_counts.values() and 2 in value_counts.values():
        return 6  # Full house
    elif 3 in value_counts.values():
        return 3  # Three of a kind
    elif list(value_counts.values()).count(2) == 2:
        return 2  # Two pair
    elif 2 in value_counts.values():
        return 1  # One pair
    else:
        # High card: use the highest card value (simplified)
        return max_card_value(values)

def max_card_value(values):
    value_map = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8,
'9': 9,
                 '10': 10, 'J': 11, 'Q': 12, 'K': 13, 'A': 14}
    return max(value_map[value] for value in values)

def apply_action(state, action):
    new_state = PokerState(state.playerHand, state.communityCards[:],
                            state.pot, state.oppBet, not state.playerTurn)
    if action == 'bet':
        new_state.pot += 10  # Example bet amount
    elif action == 'call':
        new_state.pot += state.oppBet
    elif action == 'fold':
        new_state.playerTurn = not state.playerTurn
    return new_state

def game_over(state):
    return len(state.communityCards) == 5 or state.pot >= 100  # Example conditions

def minimax(state, depth, alpha, beta, maximizing_player):
    if depth == 0 or game_over(state):
        return evaluate_state(state)

    if maximizing_player:
        max_eval = float('-inf')
        for action in generate_possible_actions(state):
            new_state = apply_action(state, action)
            eval = minimax(new_state, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float('inf')
        for action in generate_possible_actions(state):
            new_state = apply_action(state, action)
            eval = minimax(new_state, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval

# Example usage
initial_state = PokerState(playerHand=[('A', 'hearts'), ('K', 'hearts')],
                           communityCards=[('2', 'diamonds'), ('7', 'clubs'), ('Q', 'hearts')],
                           pot=20, oppBet=10, playerTurn=True)

best_action = minimax(initial_state, depth=3, alpha=float('-inf'), beta=float('inf'), maximizing_player=True)
print(f"Best action: {best_action}")
```