```
import chess
import random

# Define the Chess Environment
board = chess.Board()

# Define Rewards
WIN_REWARD = 1
LOSS_REWARD = -1
MOVE_REWARD = 0.1

# Create a Q-Table
q_table = {}

# Training Loop
num_episodes = 100
learning_rate = 0.1
discount_factor = 0.9
epsilon = 0.1

for episode in range(num_episodes):
    state = board.fen()  # Current state of the game
    total_reward = 0

    while not board.is_game_over():
        # Exploration vs. Exploitation
        if random.uniform(0, 1) < epsilon:
            # Random action
            action = random.choice(list(board.legal_moves))
        else:
            # Greedy action based on Q-values
```

```
    if state not in q_table:
        q_table[state] = {}
    legal_moves = list(board.legal_moves)
    q_values = [q_table[state].get(move.uci(), 0) for move in legal_moves]
    max_q = max(q_values, default=0)
    best_moves = [move for move, q in zip(legal_moves, q_values) if q == max_q]
    action = random.choice(best_moves)

# Take action and observe the next state and reward
move = chess.Move.from_uci(action.uci())
board.push(move)
next_state = board.fen()
reward = MOVE_REWARD

if board.is_checkmate():
    # Win the game
    reward = WIN_REWARD
elif board.is_stalemate() or board.is_insufficient_material() or board.is_seventyfive_moves():
    # Draw the game
    reward = 0

total_reward += reward

# Q-learning update
if state not in q_table:
    q_table[state] = {}
if next_state not in q_table:
    q_table[next_state] = {}

old_q = q_table[state].get(action.uci(), 0)
max_next_q = max(q_table[next_state].values(), default=0)
```

```
            new_q = (1 - learning_rate) * old_q + learning_rate * (reward + discount_factor * max_next_q)
            q_table[state][action.uci()] = new_q

            state = next_state

        # Print episode statistics
        print(f"Episode: {episode+1}, Total Reward: {total_reward}")

        # Reset the chessboard for the next episode
        board.reset()

    # Testing
    num_test_games = 10
    wins = 0

    for _ in range(num_test_games):
        board.reset()  # Reset the board before each test game
        while not board.is_game_over():
            state = board.fen()
            if state not in q_table:
                # If state is not in Q-table, choose a random move
                move = random.choice(list(board.legal_moves))
            else:
                # Choose the best move based on Q-values
                legal_moves = list(board.legal_moves)
                q_values = [q_table[state].get(move.uci(), 0) for move in legal_moves]
                max_q = max(q_values, default=0)
                best_moves = [move for move, q in zip(legal_moves, q_values) if q == max_q]
                move = random.choice(best_moves)
            move = chess.Move.from_uci(move.uci())
            board.push(move)
```

```
    if board.is_checkmate():
        wins += 1

win_rate = wins / num_test_games
print(f"Win rate against random play: {win_rate}")
```

```
Episode: 1, Total Reward: 51.30000000000046
Episode: 2, Total Reward: 40.70000000000031
Episode: 3, Total Reward: 20.600000000000023
Episode: 4, Total Reward: 32.60000000000019
Episode: 5, Total Reward: 29.400000000000148
Episode: 6, Total Reward: 40.60000000000031
Episode: 7, Total Reward: 33.500000000000206
Episode: 8, Total Reward: 32.9000000000002
Episode: 9, Total Reward: 34.800000000000225
Episode: 10, Total Reward: 44.700000000000365
Episode: 11, Total Reward: 22.300000000000047
Episode: 12, Total Reward: 49.700000000000436
Episode: 13, Total Reward: 36.60000000000025
Episode: 14, Total Reward: 33.400000000000205
Episode: 15, Total Reward: 20.100000000000016
Episode: 16, Total Reward: 39.800000000000296
Episode: 17, Total Reward: 47.600000000000406
Episode: 18, Total Reward: 38.80000000000028
Episode: 19, Total Reward: 27.800000000000125
Episode: 20, Total Reward: 43.200000000000344
Episode: 21, Total Reward: 46.9000000000004
Episode: 22, Total Reward: 50.80000000000045
Episode: 23, Total Reward: 30.100000000000158
Episode: 24, Total Reward: 40.2000000000003
Episode: 25, Total Reward: 32.30000000000019
Episode: 26, Total Reward: 24.900000000000084
Episode: 27, Total Reward: 15.099999999999966
Episode: 28, Total Reward: 34.200000000000216
Episode: 29, Total Reward: 51.20000000000046
Episode: 30, Total Reward: 49.80000000000044
Episode: 31, Total Reward: 57.70000000000055
Episode: 32, Total Reward: 26.500000000000107
Episode: 33, Total Reward: 33.70000000000021
Episode: 34, Total Reward: 40.0000000000003
Episode: 35, Total Reward: 30.000000000000156
Episode: 36, Total Reward: 38.60000000000028
Episode: 37, Total Reward: 46.40000000000039
Episode: 38, Total Reward: 32.800000000000196
Episode: 39, Total Reward: 37.000000000000256
Episode: 40, Total Reward: 33.90000000000021
Episode: 41, Total Reward: 43.00000000000034
Episode: 42, Total Reward: 42.00000000000033
Episode: 43, Total Reward: 37.90000000000027
Episode: 44, Total Reward: 67.20000000000046
Episode: 45, Total Reward: 58.100000000000556
Episode: 46, Total Reward: 37.700000000000266
Episode: 47, Total Reward: 56.50000000000053
Episode: 48, Total Reward: 58.40000000000056
Episode: 49, Total Reward: 29.700000000000152
Episode: 50, Total Reward: 53.800000000000495
Episode: 51, Total Reward: 29.700000000000152
Episode: 52, Total Reward: 42.30000000000033
Episode: 53, Total Reward: 31.800000000000182
Episode: 54, Total Reward: 32.800000000000196
Episode: 55, Total Reward: 20.900000000000013
Episode: 56, Total Reward: 31.800000000000182
Episode: 57, Total Reward: 29.800000000000153
Episode: 58, Total Reward: 11.899999999999977
```

```
Episode: 59, Total Reward: 44.30000000000036
Episode: 60, Total Reward: 29.700000000000152
Episode: 61, Total Reward: 42.500000000000334
Episode: 62, Total Reward: 71.60000000000021
Episode: 63, Total Reward: 53.00000000000048
Episode: 64, Total Reward: 26.500000000000107
Episode: 65, Total Reward: 33.80000000000021
Episode: 66, Total Reward: 42.40000000000033
Episode: 67, Total Reward: 26.200000000000102
Episode: 68, Total Reward: 24.400000000000077
Episode: 69, Total Reward: 35.20000000000023
Episode: 70, Total Reward: 49.40000000000043
Episode: 71, Total Reward: 41.40000000000032
Episode: 72, Total Reward: 29.900000000000155
Episode: 73, Total Reward: 41.00000000000031
Episode: 74, Total Reward: 36.900000000000254
Episode: 75, Total Reward: 40.80000000000031
Episode: 76, Total Reward: 35.00000000000023
Episode: 77, Total Reward: 40.3000000000003
Episode: 78, Total Reward: 44.30000000000036
Episode: 79, Total Reward: 27.800000000000125
Episode: 80, Total Reward: 38.70000000000028
Episode: 81, Total Reward: 33.60000000000021
Episode: 82, Total Reward: 18.099999999999973
Episode: 83, Total Reward: 28.200000000000117
Episode: 84, Total Reward: 26.900000000000112
Episode: 85, Total Reward: 29.10000000000013
Episode: 86, Total Reward: 37.90000000000027
Episode: 87, Total Reward: 34.900000000000226
Episode: 88, Total Reward: 32.60000000000019
Episode: 89, Total Reward: 30.100000000000158
Episode: 90, Total Reward: 14.999999999999966
Episode: 91, Total Reward: 40.2000000000003
Episode: 92, Total Reward: 46.000000000000384
Episode: 93, Total Reward: 36.50000000000025
Episode: 94, Total Reward: 42.40000000000033
Episode: 95, Total Reward: 14.09999999999997
Episode: 96, Total Reward: 19.9
Episode: 97, Total Reward: 17.29999999999996
Episode: 98, Total Reward: 33.400000000000205
Episode: 99, Total Reward: 36.80000000000025
Episode: 100, Total Reward: 38.50000000000028
Win rate against random play: 0.9
```

Aim:

The aim is to train a chess-playing agent using reinforcement learning to win games by making effective moves and learning optimal strategies through interaction with the environment.

Algorithm:

1. Initialize the Q-table to store Q-values for state-action pairs.
2. Define the hyperparameters, such as learning rate and exploration rate.
3. Training loop:

    a. Initialize the chess game board.

    b. Set the initial state of the game board.

    c. Repeat until a maximum number of episodes or convergence condition is reached:

    i. Choose an action based on the current state and exploration-exploitation trade-off.

    ii. Execute the chosen action, observe the new state and reward.

    iii. Update the Q-value of the current state-action pair using the Q-learning update equation.

    iv. Update the current state to the new state.

    d. Save the learned Q-table.
4. Testing phase:

    a. Initialize the chess game board.

    b. Set the initial state of the game board.

    c. Repeat a fixed number of test games:

    i. Choose an action based on the current state and learned Q-values.

    ii. Execute the chosen action, observe the new state and reward.

    iii. Update the current state to the new state.

    iv. Repeat until the game is over.

    d. Calculate and display the win rate against a randomly playing opponent.
5. Evaluate the agent's performance by analyzing the win rate and other metrics.

Explanation

Certainly! Here's a simplified explanation of the steps involved in implementing reinforcement learning to train a chess-playing agent:

1. Define the Chess Environment: Create a representation of the chessboard and the rules of the game.

2. Define Actions: Determine the possible moves that the agent can make based on the current state of the game.
3. Define Rewards: Design a system to provide feedback to the agent based on its actions, encouraging winning and discouraging losing.
4. Create a Q-Table: Set up a table to store the expected rewards for each state-action pair.
5. Training Loop: Repeat the following steps for multiple episodes (complete games):

a. Select an action based on the current state and the Q-table.

b. Observe the next state and the reward received after taking the action.

c. Update the Q-value for the previous state-action pair based on the observed reward and the maximum Q-value of the next state.

6. Exploration vs. Exploitation: Balance between trying out new actions and exploiting the learned knowledge in the Q-table.
7. Training Termination: Decide when to stop training, either after a specific number of episodes or when the agent reaches a desired level of performance.
8. Testing: Evaluate the performance of the trained agent by playing against human players or other chess engines.

It's important to note that for complex games like chess, traditional Q-learning with a tabular Q-table may not be feasible. Advanced methods like deep Q-learning or policy gradient methods are often used to handle the game's complexity by approximating the Q-values or policy using neural networks.

Program

```
import chess
import random

# Define the Chess Environment
board = chess.Board()

# Define Rewards
WIN_REWARD = 1
LOSS_REWARD = -1
MOVE_REWARD = 0.1

# Create a Q-Table
q_table = {}

# Training Loop
num_episodes = 10
learning_rate = 0.1
discount_factor = 0.9
epsilon = 0.1

for episode in range(num_episodes):
    state = board.fen()  # Current state of the game
    total_reward = 0

    while not board.is_game_over():
        # Exploration vs. Exploitation
        if random.uniform(0, 1) < epsilon:
            # Random action
            action = random.choice(list(board.legal_moves))
        else:
```

```
    # Greedy action based on Q-values
    if state not in q_table:
        q_table[state] = {}
    legal_moves = list(board.legal_moves)
    q_values = [q_table[state].get(move.uci(), 0) for move in legal_moves]
    max_q = max(q_values, default=0)
    best_moves = [move for move, q in zip(legal_moves, q_values) if q == max_q]
    action = random.choice(best_moves)

# Take action and observe the next state and reward
move = chess.Move.from_uci(action.uci())
board.push(move)
next_state = board.fen()
reward = MOVE_REWARD

if board.is_checkmate():
    # Win the game
    reward = WIN_REWARD
elif board.is_stalemate() or board.is_insufficient_material() or board.is_seventyfive_moves():
    # Draw the game
    reward = 0

total_reward += reward

# Q-learning update
if state not in q_table:
    q_table[state] = {}
if next_state not in q_table:
    q_table[next_state] = {}

old_q = q_table[state].get(action.uci(), 0)
```

```
        max_next_q = max(q_table[next_state].values(), default=0)
        new_q = (1 - learning_rate) * old_q + learning_rate * (reward + discount_factor * max_next_q)
        q_table[state][action.uci()] = new_q

        state = next_state

    # Print episode statistics
    print(f"Episode: {episode+1}, Total Reward: {total_reward}")

    # Reset the chessboard for the next episode
    board.reset()

# Testing
num_test_games = 10
wins = 0

for _ in range(num_test_games):
    board.reset()  # Reset the board before each test game
    while not board.is_game_over():
        state = board.fen()
        if state not in q_table:
            # If state is not in Q-table, choose a random move
            move = random.choice(list(board.legal_moves))
        else:
            # Choose the best move based on Q-values
            legal_moves = list(board.legal_moves)
            q_values = [q_table[state].get(move.uci(), 0) for move in legal_moves]
            max_q = max(q_values, default=0)
            best_moves = [move for move, q in zip(legal_moves, q_values) if q == max_q]
            move = random.choice(best_moves)
        move = chess.Move.from_uci(move.uci())
```

```
        board.push(move)

    if board.is_checkmate():
        wins += 1

win_rate = wins / num_test_games
print(f"Win rate against random play: {win_rate}")
```

Output:

```
Episode: 1, Total Reward: 25.500000000000092
Episode: 2, Total Reward: 39.20000000000029
Episode: 3, Total Reward: 37.30000000000026
Episode: 4, Total Reward: 11.09999999999998
Episode: 5, Total Reward: 38.700000000000266
Episode: 6, Total Reward: 34.00000000000021
Episode: 7, Total Reward: 41.900000000000325
Episode: 8, Total Reward: 17.399999999999963
Episode: 9, Total Reward: 37.40000000000026
Episode: 10, Total Reward: 55.80000000000052
Win rate against random play: 0.6
```

Result: