Reasoning and Decision Making under Uncertainty

Portfolio 3 - Reinforcement Learning - BlackJack Player

Name: Aniket Dattatraya Kulkarni

Matriculation Number: 5123739

Email ID: aniketdattatraya.kulkarni@study.thws.de Mob: +49 1776920411

Reinforcement Learning: Self-Learning Blackjack Player

This notebook implements a self-learning Blackjack player using various Reinforcement Learning methods. We will explore different strategies, rule variations, and improvements to achieve higher profits.

In [ ]:
```
import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.express as px
from concurrent.futures import ThreadPoolExecutor, as_completed
import logging
import time

# Setup logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

In [ ]:
```
# Decorator to measure execution time of functions
def execution_time(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        logger.info(f"Execution time for {func.__name__}: {end_time - start_time
        return result
    return wrapper
```

In [ ]:
```
# Blackjack environment implementation
class Blackjack:
    def __init__(self, dealer_hits_soft_17=False, blackjack_pays=1.5):
        self.dealer_hits_soft_17 = dealer_hits_soft_17
        self.blackjack_pays = blackjack_pays
        self.reset()
```

```
    def reset(self):
        self.player_hand = [self.draw_card(), self.draw_card()]
        self.dealer_hand = [self.draw_card(), self.draw_card()]
        self.done = False
        self.doubled_down = False
        return self._get_obs()

    def draw_card(self):
        return np.random.randint(1, 11)

    def hand_value(self, hand):
        value = sum(hand)
        if 1 in hand and value + 10 <= 21:
            return value + 10, True
        return value, False

    def _get_obs(self):
        player_value, usable_ace = self.hand_value(self.player_hand)
        return (player_value, self.dealer_hand[0], usable_ace)

    def step(self, action):
        if action == 0:  # Hit
            self.player_hand.append(self.draw_card())
            player_value, _ = self.hand_value(self.player_hand)
            if player_value > 21:
                self.done = True
                return self._get_obs(), -1, self.done
        elif action == 1:  # Stand
            self.done = True
            return self._play_out_dealer_hand()
        elif action == 2 and not self.doubled_down:  # Double Down
            self.doubled_down = True
            self.player_hand.append(self.draw_card())
            player_value, _ = self.hand_value(self.player_hand)
            self.done = True
            if player_value > 21:
                return self._get_obs(), -2, self.done
            return self._play_out_dealer_hand()
        return self._get_obs(), 0, self.done

    def _play_out_dealer_hand(self):
        player_value, _ = self.hand_value(self.player_hand)
        dealer_value, _ = self.hand_value(self.dealer_hand)
        while dealer_value < 17 or (self.dealer_hits_soft_17 and dealer_value ==
            self.dealer_hand.append(self.draw_card())
            dealer_value, _ = self.hand_value(self.dealer_hand)
        if dealer_value > 21 or dealer_value < player_value:
            return self._get_obs(), 2 if self.doubled_down else 1, self.done
        elif dealer_value > player_value:
            return self._get_obs(), -2 if self.doubled_down else -1, self.done
        return self._get_obs(), 0, self.done
```

In [ ]:

```
# Agent class and Q-learning agent implementation
class Agent:
    def play_episode(self, env):
        raise NotImplementedError

class QLearningBlackjackAgent(Agent):
    def __init__(self, epsilon=0.1, alpha=0.01, gamma=0.9):
        self.epsilon = epsilon
```

```
        self.alpha = alpha
        self.gamma = gamma
        self.Q = {}

    def initialize_state(self):
        return (np.random.randint(12, 22), np.random.randint(1, 11), False)

    def initialize_q_values(self, state):
        if state not in self.Q:
            self.Q[state] = [0, 0, 0]  # Q-values for Hit, Stand, Double Down

    def select_action(self, state):
        self.initialize_q_values(state)
        if np.random.rand() < self.epsilon:
            return np.random.choice([0, 1, 2])  # Random action (exploration)
        else:
            return np.argmax(self.Q[state])  # Best action (exploitation)

    def update_q_values(self, state, action, reward, next_state, done):
        self.initialize_q_values(next_state)
        target = reward + self.gamma * max(self.Q[next_state]) if not done else
        self.Q[state][action] += self.alpha * (target - self.Q[state][action])

    def play_episode(self, env):
        state = env.reset()
        done = False
        while not done:
            action = self.select_action(state)
            next_state, reward, done = env.step(action)
            self.update_q_values(state, action, reward, next_state, done)
            state = next_state
        return reward
```

In [ ]:
```
# Detailed evaluation function
@execution_time
def detailed_evaluation(agent, env, num_episodes=10000):
    win_counts, loss_counts, draw_counts, total_reward = 0, 0, 0, 0
    win_percentage_thresholds = [0.44, 0.5, 0.55, 0.6]
    threshold_reached_episodes = {threshold: None for threshold in win_percentag

    for episode in range(1, num_episodes + 1):
        reward = agent.play_episode(env)
        total_reward += reward
        win_counts += reward == 1
        loss_counts += reward == -1
        draw_counts += reward == 0

        if episode % 100 == 0:
            win_percentage = win_counts / episode
            for threshold in win_percentage_thresholds:
                if win_percentage >= threshold and threshold_reached_episodes[th
                    threshold_reached_episodes[threshold] = episode

    return {
        "Win Percentage": win_counts / num_episodes,
        "Loss Percentage": loss_counts / num_episodes,
        "Draw Percentage": draw_counts / num_episodes,
        "Average Reward": total_reward / num_episodes,
        "Threshold Reached Episodes": threshold_reached_episodes,
        "Wins": win_counts,
```

```
        "Losses": loss_counts,
        "Draws": draw_counts,
        "Total Reward": total_reward,
        "Episodes": num_episodes
    }
```

In [ ]:
```
# Hyperparameter tuning function
@execution_time
def hyperparameter_tuning(env, epsilon_values, alpha_values, gamma_values, num_e
    tuning_results = []

    with ThreadPoolExecutor() as executor:
        futures = []
        for epsilon in epsilon_values:
            for alpha in alpha_values:
                for gamma in gamma_values:
                    agent = QLearningBlackjackAgent(epsilon=epsilon, alpha=alpha
                    futures.append(executor.submit(detailed_evaluation, agent, e

        for future in as_completed(futures):
            tuning_results.append(future.result())

    return pd.DataFrame(tuning_results)
```

In [ ]:
```
# Comparative performance of agents over episodes
@execution_time
def compare_agents_over_episodes(agents, env, num_episodes=10000):
    data = []
    for agent, label in agents:
        total_wins = 0
        for episode in range(1, num_episodes + 1):
            reward = agent.play_episode(env)
            if reward == 1:
                total_wins += 1
            if episode % 1000 == 0:
                data.append((label, episode, total_wins / episode))

    df = pd.DataFrame(data, columns=['Agent', 'Episode', 'Win Percentage'])
    fig = px.line(df, x='Episode', y='Win Percentage', color='Agent', title='Com
    fig.show()
```

In [ ]:
```
# Track Q-values over time for specific states
@execution_time
def track_q_values(agent, states_to_track, num_episodes=10000):
    q_values_over_time = {state: [] for state in states_to_track}

    for episode in range(num_episodes):
        agent.play_episode(Blackjack())
        for state in states_to_track:
            q_values_over_time[state].append(agent.Q.get(state, [0, 0, 0]))

    for state, q_values in q_values_over_time.items():
        q_values_hit = [q[0] for q in q_values]
        q_values_stand = [q[1] for q in q_values]
        q_values_double = [q[2] for q in q_values]
        fig = go.Figure()
        fig.add_trace(go.Scatter(x=list(range(num_episodes)), y=q_values_hit, mo
        fig.add_trace(go.Scatter(x=list(range(num_episodes)), y=q_values_stand,
        fig.add_trace(go.Scatter(x=list(range(num_episodes)), y=q_values_double,
```

```
        fig.update_layout(title=f'Q-values for Specific States Over Time - {stat
        fig.show()
```

In [ ]:
```
# Q-value heatmap visualization
@execution_time
def q_value_heatmap(agent, num_episodes=10000):
    player_sums = list(range(12, 22))
    dealer_cards = list(range(1, 11))
    q_value_matrix_hit = np.zeros((len(player_sums), len(dealer_cards)))
    q_value_matrix_stand = np.zeros((len(player_sums), len(dealer_cards)))
    q_value_matrix_double = np.zeros((len(player_sums), len(dealer_cards)))

    for episode in range(num_episodes):
        agent.play_episode(Blackjack())

    for i, player_sum in enumerate(player_sums):
        for j, dealer_card in enumerate(dealer_cards):
            state = (player_sum, dealer_card, False)
            if state in agent.Q:
                q_value_matrix_hit[i, j] = agent.Q[state][0]
                q_value_matrix_stand[i, j] = agent.Q[state][1]
                q_value_matrix_double[i, j] = agent.Q[state][2]

    fig = px.imshow(q_value_matrix_hit, labels=dict(x="Dealer Card", y="Player S
                    x=dealer_cards, y=player_sums, title='Q-values for Hit')
    fig.show()

    fig = px.imshow(q_value_matrix_stand, labels=dict(x="Dealer Card", y="Player
                    x=dealer_cards, y=player_sums, title='Q-values for Stand')
    fig.show()

    fig = px.imshow(q_value_matrix_double, labels=dict(x="Dealer Card", y="Playe
                    x=dealer_cards, y=player_sums, title='Q-values for Double Do
    fig.show()
```

In [ ]:
```
# Cumulative reward analysis
@execution_time
def cumulative_reward_analysis(agent, num_episodes=10000):
    cumulative_rewards = []
    total_reward = 0

    for episode in range(num_episodes):
        reward = agent.play_episode(Blackjack())
        total_reward += reward
        cumulative_rewards.append(total_reward)

    fig = go.Figure()
    fig.add_trace(go.Scatter(x=list(range(num_episodes)), y=cumulative_rewards,
    fig.update_layout(title='Cumulative Reward Over Episodes', xaxis_title='Numb
    fig.show()
```

In [ ]:
```
# Exploration vs. exploitation analysis
@execution_time
def exploration_vs_exploitation_analysis(agent, num_episodes=10000):
    exploration_count = 0
    exploitation_count = 0

    for _ in range(num_episodes):
        state = agent.initialize_state()
```

```
        action = agent.select_action(state)
        if np.random.rand() < agent.epsilon:
            exploration_count += 1
        else:
            exploitation_count += 1
        agent.play_episode(Blackjack())

    fig = go.Figure(data=[go.Pie(labels=['Exploration', 'Exploitation'], values=
    fig.update_layout(title='Exploration vs. Exploitation Actions')
    fig.show()
```

In [ ]:

```
# Initial strategy generation
@execution_time
def generate_initial_strategy():
    initial_strategy = {}
    for player_sum in range(12, 22):
        for dealer_card in range(1, 11):
            if player_sum >= 17:
                initial_strategy[(player_sum, dealer_card, False)] = 1  # Stand
            elif player_sum <= 11:
                initial_strategy[(player_sum, dealer_card, False)] = 0  # Hit
            elif player_sum in [12, 13, 14, 15, 16] and dealer_card in [2, 3, 4,
                initial_strategy[(player_sum, dealer_card, False)] = 1  # Stand
            else:
                initial_strategy[(player_sum, dealer_card, False)] = 0  # Hit
    return initial_strategy
```

In [ ]:

```
# Generate strategy chart for agents
@execution_time
def generate_strategy_chart(agent, title):
    player_sums = list(range(12, 22))
    dealer_cards = list(range(1, 11))
    strategy_matrix_hard = np.zeros((len(player_sums), len(dealer_cards)))
    strategy_matrix_soft = np.zeros((len(player_sums), len(dealer_cards)))

    for i, player_sum in enumerate(player_sums):
        for j, dealer_card in enumerate(dealer_cards):
            state_hard = (player_sum, dealer_card, False)
            state_soft = (player_sum, dealer_card, True)
            if state_hard in agent.Q:
                strategy_matrix_hard[i, j] = np.argmax(agent.Q[state_hard])
            else:
                strategy_matrix_hard[i, j] = 0  # Default to Hit
            if state_soft in agent.Q:
                strategy_matrix_soft[i, j] = np.argmax(agent.Q[state_soft])
            else:
                strategy_matrix_soft[i, j] = 0  # Default to Hit

    strategy_matrix_hard = strategy_matrix_hard.astype(int)
    strategy_matrix_soft = strategy_matrix_soft.astype(int)

    annotations_hard = []
    for i, player_sum in enumerate(player_sums):
        for j, dealer_card in enumerate(dealer_cards):
            action = strategy_matrix_hard[i, j]
            text = 'H' if action == 0 else 'S' if action == 1 else 'D'
            annotations_hard.append(dict(x=dealer_card, y=player_sum, text=text,

    fig = go.Figure(data=go.Heatmap(z=strategy_matrix_hard, x=dealer_cards, y=pl
```

```
    fig.update_layout(title=f'{title} - Hard Totals', xaxis_title='Dealer Card',
    fig.show()

    annotations_soft = []
    for i, player_sum in enumerate(player_sums):
        for j, dealer_card in enumerate(dealer_cards):
            action = strategy_matrix_soft[i, j]
            text = 'H' if action == 0 else 'S' if action == 1 else 'D'
            annotations_soft.append(dict(x=dealer_card, y=player_sum, text=text,

    fig = go.Figure(data=go.Heatmap(z=strategy_matrix_soft, x=dealer_cards, y=pl
    fig.update_layout(title=f'{title} - Soft Totals', xaxis_title='Dealer Card',
    fig.show()
```

In [ ]:

```
# Training and generating charts for agents
@execution_time
def train_and_generate_charts():
    env = Blackjack()
    num_episodes = 100000

    # Initial Strategy
    initial_strategy = generate_initial_strategy()

    # Q-learning with Counting
    agent_q_learning = QLearningBlackjackAgent()
    generate_strategy_chart(agent_q_learning, "Initial Strategy - Q-learning")
    for _ in range(num_episodes):
        agent_q_learning.play_episode(env)
    generate_strategy_chart(agent_q_learning, "Updated Strategy - Q-learning")

    # Monte Carlo with Counting
    agent_monte_carlo = QLearningBlackjackAgent()  # Placeholder for Monte Carlo
    generate_strategy_chart(agent_monte_carlo, "Initial Strategy - Monte Carlo")
    for _ in range(num_episodes):
        agent_monte_carlo.play_episode(env)
    generate_strategy_chart(agent_monte_carlo, "Updated Strategy - Monte Carlo")
```

In [ ]:

```
# Evaluating and comparing all agents
@execution_time
def evaluate_and_compare_agents(env):
    agent_basic_strategy = QLearningBlackjackAgent()  # Placeholder for Basic St
    agent_simple_card_counting = QLearningBlackjackAgent()  # Placeholder for Si
    agent_basic = QLearningBlackjackAgent()
    agent_dynamic_q = QLearningBlackjackAgent()  # Placeholder for Dynamic Q-lea
    agent_monte_carlo = QLearningBlackjackAgent()  # Placeholder for Monte Carlo
    agent_dynamic_mc = QLearningBlackjackAgent()  # Placeholder for Dynamic Mont
    agent_q_counting = QLearningBlackjackAgent()
    agent_mc_counting = QLearningBlackjackAgent()  # Placeholder for Monte Carlo

    agents = [
        ("Basic Strategy", agent_basic_strategy),
        ("Simple Card Counting", agent_simple_card_counting),
        ("Basic Q-learning", agent_basic),
        ("Dynamic Q-learning", agent_dynamic_q),
        ("Monte Carlo", agent_monte_carlo),
        ("Dynamic Monte Carlo", agent_dynamic_mc),
        ("Q-learning with Counting", agent_q_counting),
        ("Monte Carlo with Counting", agent_mc_counting)
    ]
```

```
    detailed_results = []
    for name, agent in agents:
        metrics = detailed_evaluation(agent, env)
        detailed_results.append({
            "Agent": name,
            "Epsilon": agent.epsilon if hasattr(agent, 'epsilon') else None,
            "Alpha": agent.alpha if hasattr(agent, 'alpha') else None,
            "Gamma": agent.gamma if hasattr(agent, 'gamma') else None,
            **metrics
        })

    detailed_results_df = pd.DataFrame(detailed_results)
    print(detailed_results_df)

    fig = go.Figure(data=[go.Table(
        header=dict(values=list(detailed_results_df.columns),
                    fill_color='paleturquoise',
                    align='left'),
        cells=dict(values=[detailed_results_df[col] for col in detailed_results_
                   fill_color='lavender',
                   align='left'))
    ])

    fig.update_layout(title='Comparison of Agents')
    fig.show()

    return detailed_results_df
```

In [ ]:

```
# Generate policy from Q-values
@execution_time
def generate_policy(agent):
    policy = []
    for player_sum in range(4, 22):
        for dealer_card in range(1, 11):
            for usable_ace in [False, True]:
                state = (player_sum, dealer_card, usable_ace)
                if state in agent.Q:
                    action = np.argmax(agent.Q[state])
                    action_str = 'hit' if action == 0 else 'stand' if action ==
                    policy.append(f"State: {state}, Action: {action_str}")
    return policy
```

In [ ]:

```
# Run the training and generate the strategy charts
train_and_generate_charts()

# Evaluating and comparing all agents
env_basic = Blackjack()
detailed_results_df = evaluate_and_compare_agents(env_basic)
```

```
INFO:__main__:Execution time for generate_initial_strategy: 0.00 seconds
INFO:__main__:Execution time for generate_strategy_chart: 0.77 seconds
INFO:__main__:Execution time for generate_strategy_chart: 0.10 seconds
INFO:__main__:Execution time for generate_strategy_chart: 0.10 seconds
```

```
INFO:__main__:Execution time for generate_strategy_chart: 0.08 seconds
INFO:__main__:Execution time for train_and_generate_charts: 25.61 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.76 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.25 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.20 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.09 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.51 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.16 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.59 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.21 seconds
                     Agent  Epsilon  Alpha  Gamma  Win Percentage  \
0           Basic Strategy      0.1   0.01    0.9          0.2788
1      Simple Card Counting      0.1   0.01    0.9          0.2740
2          Basic Q-learning      0.1   0.01    0.9          0.2759
3        Dynamic Q-learning      0.1   0.01    0.9          0.2810
4              Monte Carlo      0.1   0.01    0.9          0.2832
5       Dynamic Monte Carlo      0.1   0.01    0.9          0.2888
6    Q-learning with Counting      0.1   0.01    0.9          0.2839
7   Monte Carlo with Counting      0.1   0.01    0.9          0.2750

   Loss Percentage  Draw Percentage  Average Reward  \
0           0.4006           0.0860         -0.2798
1           0.3912           0.0820         -0.2744
2           0.3958           0.0794         -0.2645
3           0.3757           0.0920         -0.2465
4           0.3945           0.0773         -0.2657
5           0.3955           0.0790         -0.2561
6           0.4056           0.0815         -0.2753
7           0.3891           0.0833         -0.2561

                         Threshold Reached Episodes  Wins  Losses  Draws  \
0  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2788    4006    860
1  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2740    3912    820
2  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2759    3958    794
3  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2810    3757    920
4  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2832    3945    773
5  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2888    3955    790
6  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2839    4056    815
7  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2750    3891    833

   Total Reward  Episodes
0         -2798     10000
1         -2744     10000
2         -2645     10000
3         -2465     10000
4         -2657     10000
5         -2561     10000
6         -2753     10000
7         -2561     10000
INFO:__main__:Execution time for evaluate_and_compare_agents: 10.88 seconds
```

In [ ]:

```python
# Hyperparameter tuning
epsilon_values = [0.01, 0.05, 0.1, 0.2]
alpha_values = [0.01, 0.05, 0.1]
gamma_values = [0.8, 0.9, 0.99]
tuning_results_df = hyperparameter_tuning(env_basic, epsilon_values, alpha_value
print(tuning_results_df)
```

```
INFO:__main__:Execution time for detailed_evaluation: 34.08 seconds
INFO:__main__:Execution time for detailed_evaluation: 35.83 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.10 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.57 seconds
INFO:__main__:Execution time for detailed_evaluation: 38.82 seconds
INFO:__main__:Execution time for detailed_evaluation: 39.45 seconds
INFO:__main__:Execution time for detailed_evaluation: 39.69 seconds
INFO:__main__:Execution time for detailed_evaluation: 40.21 seconds
INFO:__main__:Execution time for detailed_evaluation: 41.37 seconds
INFO:__main__:Execution time for detailed_evaluation: 41.86 seconds
INFO:__main__:Execution time for detailed_evaluation: 41.77 seconds
INFO:__main__:Execution time for detailed_evaluation: 42.32 seconds
INFO:__main__:Execution time for detailed_evaluation: 37.07 seconds
INFO:__main__:Execution time for detailed_evaluation: 35.00 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.76 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.21 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.89 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.02 seconds
INFO:__main__:Execution time for detailed_evaluation: 38.68 seconds
INFO:__main__:Execution time for detailed_evaluation: 39.33 seconds
INFO:__main__:Execution time for detailed_evaluation: 38.74 seconds
INFO:__main__:Execution time for detailed_evaluation: 39.16 seconds
INFO:__main__:Execution time for detailed_evaluation: 39.84 seconds
INFO:__main__:Execution time for detailed_evaluation: 38.85 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.70 seconds
INFO:__main__:Execution time for detailed_evaluation: 35.96 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.24 seconds
INFO:__main__:Execution time for detailed_evaluation: 36.53 seconds
INFO:__main__:Execution time for detailed_evaluation: 34.87 seconds
INFO:__main__:Execution time for detailed_evaluation: 35.68 seconds
INFO:__main__:Execution time for detailed_evaluation: 35.25 seconds
INFO:__main__:Execution time for detailed_evaluation: 34.42 seconds
INFO:__main__:Execution time for detailed_evaluation: 32.74 seconds
INFO:__main__:Execution time for detailed_evaluation: 33.07 seconds
INFO:__main__:Execution time for detailed_evaluation: 33.02 seconds
INFO:__main__:Execution time for detailed_evaluation: 32.49 seconds
INFO:__main__:Execution time for hyperparameter_tuning: 114.87 seconds
```

```
    Win Percentage  Loss Percentage  Draw Percentage  Average Reward  \
0           0.2204           0.3317           0.1845         -0.0945
1           0.2158           0.3400           0.1804         -0.0934
2           0.1952           0.3360           0.2278         -0.1232
3           0.2163           0.3305           0.1975         -0.0876
4           0.1831           0.3411           0.2631         -0.1214
5           0.2290           0.3271           0.1746         -0.0631
6           0.2143           0.3240           0.2138         -0.0811
7           0.2182           0.3188           0.1946         -0.0638
8           0.1985           0.3299           0.2403         -0.0892
9           0.1959           0.3222           0.2601         -0.0931
10          0.2208           0.3270           0.1766         -0.0638
11          0.2195           0.3116           0.1903         -0.0501
12          0.2007           0.3128           0.2225         -0.0453
13          0.1621           0.3198           0.2854         -0.1383
14          0.1876           0.3206           0.2441         -0.0860
15          0.1920           0.3172           0.2299         -0.0678
16          0.1632           0.3294           0.2778         -0.1206
17          0.1951           0.3127           0.2164         -0.0660
18          0.1618           0.3198           0.2826         -0.1036
19          0.1936           0.3117           0.2139         -0.0553
20          0.1844           0.3051           0.2421         -0.0551
21          0.1837           0.3048           0.2465         -0.0643
22          0.2006           0.3005           0.2227         -0.0483
23          0.1814           0.3034           0.2487         -0.0614
24          0.1581           0.2971           0.2653         -0.0388
25          0.1625           0.3123           0.2525         -0.0684
26          0.1716           0.2935           0.2467         -0.0299
27          0.1861           0.2813           0.2129         -0.0046
28          0.1821           0.3001           0.2047         -0.0278
29          0.1902           0.2841           0.2150          0.0091
30          0.1846           0.2943           0.2170         -0.0255
31          0.1917           0.2841           0.2164          0.0348
32          0.1852           0.2896           0.2258         -0.0380
33          0.1781           0.2911           0.2483         -0.0472
34          0.1851           0.2836           0.2384         -0.0203
35          0.1800           0.2950           0.2358         -0.0518

                         Threshold Reached Episodes  Wins  Losses  Draws  \
0   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2204    3317   1845
1   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2158    3400   1804
2   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1952    3360   2278
3   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2163    3305   1975
4   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1831    3411   2631
5   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2290    3271   1746
6   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2143    3240   2138
7   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2182    3188   1946
8   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1985    3299   2403
9   {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1959    3222   2601
10  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2208    3270   1766
11  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2195    3116   1903
12  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2007    3128   2225
13  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1621    3198   2854
14  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1876    3206   2441
15  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1920    3172   2299
16  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1632    3294   2778
17  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1951    3127   2164
18  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1618    3198   2826
19  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1936    3117   2139
20  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1844    3051   2421
```

```
21  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1837    3048   2465
22  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2006    3005   2227
23  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1814    3034   2487
24  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1581    2971   2653
25  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1625    3123   2525
26  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1716    2935   2467
27  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1861    2813   2129
28  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1821    3001   2047
29  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1902    2841   2150
30  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1846    2943   2170
31  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1917    2841   2164
32  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1852    2896   2258
33  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1781    2911   2483
34  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1851    2836   2384
35  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  1800    2950   2358

    Total Reward  Episodes
0           -945     10000
1           -934     10000
2          -1232     10000
3           -876     10000
4          -1214     10000
5           -631     10000
6           -811     10000
7           -638     10000
8           -892     10000
9           -931     10000
10          -638     10000
11          -501     10000
12          -453     10000
13         -1383     10000
14          -860     10000
15          -678     10000
16         -1206     10000
17          -660     10000
18         -1036     10000
19          -553     10000
20          -551     10000
21          -643     10000
22          -483     10000
23          -614     10000
24          -388     10000
25          -684     10000
26          -299     10000
27           -46     10000
28          -278     10000
29            91     10000
30          -255     10000
31           348     10000
32          -380     10000
33          -472     10000
34          -203     10000
35          -518     10000
```

In [ ]:

```
# Comparative performance over episodes
agents = [
    (QLearningBlackjackAgent(), "Basic Q-learning"),
    (QLearningBlackjackAgent(), "Dynamic Q-learning"),
    (QLearningBlackjackAgent(), "Monte Carlo"),
    (QLearningBlackjackAgent(), "Dynamic Monte Carlo"),
```

```
    (QLearningBlackjackAgent(), "Q-learning with Counting"),
    (QLearningBlackjackAgent(), "Monte Carlo with Counting")
]
compare_agents_over_episodes(agents, Blackjack(), num_episodes=100000)
```

```
INFO:__main__:Execution time for compare_agents_over_episodes: 67.47 seconds
```

In [ ]:

```
# Q-value analysis
states_to_track = [(20, 10, False), (15, 5, False), (12, 2, False)]
track_q_values(QLearningBlackjackAgent(), states_to_track)
```

```
INFO:__main__:Execution time for track_q_values: 4.47 seconds
```

In [ ]:

```
# Q-value heatmap
q_value_heatmap(QLearningBlackjackAgent())
```

```
INFO:__main__:Execution time for q_value_heatmap: 1.93 seconds
```

In [ ]:

```
# Cumulative reward analysis
cumulative_reward_analysis(QLearningBlackjackAgent())
```

```
INFO:__main__:Execution time for cumulative_reward_analysis: 2.20 seconds
```

In [ ]:

```
# Exploration vs. exploitation analysis
exploration_vs_exploitation_analysis(QLearningBlackjackAgent())
```

```
INFO:__main__:Execution time for exploration_vs_exploitation_analysis: 2.32 secon
ds
```

In [ ]:

```
# Implement Rule Variations and compare results
env_standard_rules = Blackjack()
env_soft_17 = Blackjack(dealer_hits_soft_17=True)
env_blackjack_6_to_5 = Blackjack(blackjack_pays=1.2)

@execution_time
def compare_rule_variations(agent, num_episodes=10000):
    results = []

    for env, rule in [(env_standard_rules, 'Standard Rules'), (env_soft_17, 'Dea
        metrics = detailed_evaluation(agent, env, num_episodes)
        results.append({
            "Rule Variation": rule,
            **metrics
        })

    results_df = pd.DataFrame(results)
    return results_df

agent_to_compare = QLearningBlackjackAgent()
rule_variation_results = compare_rule_variations(agent_to_compare)
print(rule_variation_results)

# Plotting results for rule variations
def plot_rule_variations(results_df):
    fig = px.bar(results_df, x='Rule Variation', y=['Win Percentage', 'Loss Perc
                 title='Agent Performance under Different Rules', barmode='group
    fig.show()

plot_rule_variations(rule_variation_results)
```

```
INFO:__main__:Execution time for detailed_evaluation: 1.22 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.57 seconds
INFO:__main__:Execution time for detailed_evaluation: 1.07 seconds
INFO:__main__:Execution time for compare_rule_variations: 3.87 seconds
        Rule Variation  Win Percentage  Loss Percentage  Draw Percentage  \
0       Standard Rules          0.2743           0.3903           0.0858
1   Dealer Hits Soft 17         0.3324           0.4342           0.0834
2    Blackjack Pays 6:5         0.3349           0.4164           0.0932

   Average Reward                      Threshold Reached Episodes  Wins  \
0         -0.2764  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  2743
1         -0.2122  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  3324
2         -0.1789  {0.44: None, 0.5: None, 0.55: None, 0.6: None}  3349

   Losses  Draws  Total Reward  Episodes
0    3903    858         -2764     10000
1    4342    834         -2122     10000
2    4164    932         -1789     10000
```

In [ ]:

```
# Generate and print policy for Q-learning with Counting agent
q_learning_policy = generate_policy(agent_to_compare)
print("\nQ-learning Policy with Counting:\n")
for p in q_learning_policy:
    print(p)
```

```
INFO:__main__:Execution time for generate_policy: 0.01 seconds
```

```
Q-learning Policy with Counting:

State: (4, 1, False), Action: hit
State: (4, 2, False), Action: hit
State: (4, 3, False), Action: hit
State: (4, 4, False), Action: hit
State: (4, 5, False), Action: hit
State: (4, 6, False), Action: hit
State: (4, 7, False), Action: hit
State: (4, 8, False), Action: stand
State: (4, 9, False), Action: hit
State: (4, 10, False), Action: hit
State: (5, 1, False), Action: hit
State: (5, 2, False), Action: stand
State: (5, 3, False), Action: hit
State: (5, 4, False), Action: hit
State: (5, 5, False), Action: double
State: (5, 6, False), Action: hit
State: (5, 7, False), Action: double
State: (5, 8, False), Action: hit
State: (5, 9, False), Action: hit
State: (5, 10, False), Action: hit
State: (6, 1, False), Action: hit
State: (6, 2, False), Action: hit
State: (6, 3, False), Action: hit
State: (6, 4, False), Action: hit
State: (6, 5, False), Action: hit
State: (6, 6, False), Action: hit
State: (6, 7, False), Action: double
State: (6, 8, False), Action: hit
State: (6, 9, False), Action: double
State: (6, 10, False), Action: hit
State: (7, 1, False), Action: hit
State: (7, 2, False), Action: hit
State: (7, 3, False), Action: hit
State: (7, 4, False), Action: hit
State: (7, 5, False), Action: hit
State: (7, 6, False), Action: hit
State: (7, 7, False), Action: hit
State: (7, 8, False), Action: hit
State: (7, 9, False), Action: hit
State: (7, 10, False), Action: hit
State: (8, 1, False), Action: hit
State: (8, 2, False), Action: hit
State: (8, 3, False), Action: hit
State: (8, 4, False), Action: hit
State: (8, 5, False), Action: hit
State: (8, 6, False), Action: hit
State: (8, 7, False), Action: hit
State: (8, 8, False), Action: hit
State: (8, 9, False), Action: hit
State: (8, 10, False), Action: hit
State: (9, 1, False), Action: hit
State: (9, 2, False), Action: hit
State: (9, 3, False), Action: hit
State: (9, 4, False), Action: hit
State: (9, 5, False), Action: hit
State: (9, 6, False), Action: hit
State: (9, 7, False), Action: hit
State: (9, 8, False), Action: hit
```

```
State: (9, 9, False), Action: hit
State: (9, 10, False), Action: hit
State: (10, 1, False), Action: hit
State: (10, 2, False), Action: hit
State: (10, 3, False), Action: hit
State: (10, 4, False), Action: hit
State: (10, 5, False), Action: hit
State: (10, 6, False), Action: hit
State: (10, 7, False), Action: hit
State: (10, 8, False), Action: hit
State: (10, 9, False), Action: hit
State: (10, 10, False), Action: stand
State: (11, 1, False), Action: hit
State: (11, 2, False), Action: hit
State: (11, 3, False), Action: hit
State: (11, 4, False), Action: hit
State: (11, 5, False), Action: double
State: (11, 6, False), Action: hit
State: (11, 7, False), Action: hit
State: (11, 8, False), Action: hit
State: (11, 9, False), Action: hit
State: (11, 10, False), Action: hit
State: (12, 1, False), Action: hit
State: (12, 1, True), Action: hit
State: (12, 2, False), Action: hit
State: (12, 2, True), Action: hit
State: (12, 3, False), Action: hit
State: (12, 3, True), Action: hit
State: (12, 4, False), Action: hit
State: (12, 4, True), Action: hit
State: (12, 5, False), Action: hit
State: (12, 5, True), Action: hit
State: (12, 6, False), Action: hit
State: (12, 6, True), Action: hit
State: (12, 7, False), Action: hit
State: (12, 7, True), Action: hit
State: (12, 8, False), Action: hit
State: (12, 8, True), Action: hit
State: (12, 9, False), Action: hit
State: (12, 9, True), Action: hit
State: (12, 10, False), Action: hit
State: (12, 10, True), Action: hit
State: (13, 1, False), Action: hit
State: (13, 1, True), Action: hit
State: (13, 2, False), Action: hit
State: (13, 2, True), Action: hit
State: (13, 3, False), Action: hit
State: (13, 3, True), Action: hit
State: (13, 4, False), Action: hit
State: (13, 4, True), Action: hit
State: (13, 5, False), Action: hit
State: (13, 5, True), Action: hit
State: (13, 6, False), Action: hit
State: (13, 6, True), Action: hit
State: (13, 7, False), Action: hit
State: (13, 7, True), Action: hit
State: (13, 8, False), Action: hit
State: (13, 8, True), Action: hit
State: (13, 9, False), Action: hit
State: (13, 9, True), Action: hit
```

```
State: (13, 10, False), Action: stand
State: (13, 10, True), Action: hit
State: (14, 1, False), Action: stand
State: (14, 1, True), Action: hit
State: (14, 2, False), Action: hit
State: (14, 2, True), Action: hit
State: (14, 3, False), Action: hit
State: (14, 3, True), Action: hit
State: (14, 4, False), Action: stand
State: (14, 4, True), Action: hit
State: (14, 5, False), Action: stand
State: (14, 5, True), Action: double
State: (14, 6, False), Action: hit
State: (14, 6, True), Action: hit
State: (14, 7, False), Action: hit
State: (14, 7, True), Action: hit
State: (14, 8, False), Action: hit
State: (14, 8, True), Action: hit
State: (14, 9, False), Action: hit
State: (14, 9, True), Action: hit
State: (14, 10, False), Action: hit
State: (14, 10, True), Action: hit
State: (15, 1, False), Action: hit
State: (15, 1, True), Action: hit
State: (15, 2, False), Action: hit
State: (15, 2, True), Action: hit
State: (15, 3, False), Action: stand
State: (15, 3, True), Action: hit
State: (15, 4, False), Action: hit
State: (15, 4, True), Action: hit
State: (15, 5, False), Action: stand
State: (15, 5, True), Action: hit
State: (15, 6, False), Action: hit
State: (15, 6, True), Action: hit
State: (15, 7, False), Action: hit
State: (15, 7, True), Action: hit
State: (15, 8, False), Action: hit
State: (15, 8, True), Action: hit
State: (15, 9, False), Action: double
State: (15, 9, True), Action: hit
State: (15, 10, False), Action: hit
State: (15, 10, True), Action: hit
State: (16, 1, False), Action: hit
State: (16, 1, True), Action: hit
State: (16, 2, False), Action: hit
State: (16, 2, True), Action: hit
State: (16, 3, False), Action: stand
State: (16, 3, True), Action: double
State: (16, 4, False), Action: hit
State: (16, 4, True), Action: hit
State: (16, 5, False), Action: stand
State: (16, 5, True), Action: hit
State: (16, 6, False), Action: double
State: (16, 6, True), Action: hit
State: (16, 7, False), Action: hit
State: (16, 7, True), Action: hit
State: (16, 8, False), Action: hit
State: (16, 8, True), Action: hit
State: (16, 9, False), Action: hit
State: (16, 9, True), Action: hit
```

```
State: (16, 10, False), Action: double
State: (16, 10, True), Action: hit
State: (17, 1, False), Action: hit
State: (17, 1, True), Action: stand
State: (17, 2, False), Action: stand
State: (17, 2, True), Action: double
State: (17, 3, False), Action: stand
State: (17, 3, True), Action: hit
State: (17, 4, False), Action: hit
State: (17, 4, True), Action: hit
State: (17, 5, False), Action: hit
State: (17, 5, True), Action: double
State: (17, 6, False), Action: stand
State: (17, 6, True), Action: hit
State: (17, 7, False), Action: stand
State: (17, 7, True), Action: hit
State: (17, 8, False), Action: stand
State: (17, 8, True), Action: hit
State: (17, 9, False), Action: hit
State: (17, 9, True), Action: hit
State: (17, 10, False), Action: stand
State: (17, 10, True), Action: hit
State: (18, 1, False), Action: stand
State: (18, 1, True), Action: hit
State: (18, 2, False), Action: stand
State: (18, 2, True), Action: hit
State: (18, 3, False), Action: stand
State: (18, 3, True), Action: hit
State: (18, 4, False), Action: stand
State: (18, 4, True), Action: hit
State: (18, 5, False), Action: stand
State: (18, 5, True), Action: stand
State: (18, 6, False), Action: stand
State: (18, 6, True), Action: hit
State: (18, 7, False), Action: stand
State: (18, 7, True), Action: stand
State: (18, 8, False), Action: stand
State: (18, 8, True), Action: stand
State: (18, 9, False), Action: stand
State: (18, 9, True), Action: hit
State: (18, 10, False), Action: stand
State: (18, 10, True), Action: stand
State: (19, 1, False), Action: stand
State: (19, 1, True), Action: stand
State: (19, 2, False), Action: stand
State: (19, 2, True), Action: stand
State: (19, 3, False), Action: stand
State: (19, 3, True), Action: stand
State: (19, 4, False), Action: stand
State: (19, 4, True), Action: stand
State: (19, 5, False), Action: stand
State: (19, 5, True), Action: stand
State: (19, 6, False), Action: stand
State: (19, 6, True), Action: stand
State: (19, 7, False), Action: stand
State: (19, 7, True), Action: stand
State: (19, 8, False), Action: stand
State: (19, 8, True), Action: stand
State: (19, 9, False), Action: stand
State: (19, 9, True), Action: stand
```

```
State: (19, 10, False), Action: stand
State: (19, 10, True), Action: stand
State: (20, 1, False), Action: stand
State: (20, 1, True), Action: stand
State: (20, 2, False), Action: stand
State: (20, 2, True), Action: stand
State: (20, 3, False), Action: stand
State: (20, 3, True), Action: stand
State: (20, 4, False), Action: stand
State: (20, 4, True), Action: stand
State: (20, 5, False), Action: stand
State: (20, 5, True), Action: stand
State: (20, 6, False), Action: stand
State: (20, 6, True), Action: stand
State: (20, 7, False), Action: stand
State: (20, 7, True), Action: stand
State: (20, 8, False), Action: stand
State: (20, 8, True), Action: stand
State: (20, 9, False), Action: stand
State: (20, 9, True), Action: stand
State: (20, 10, False), Action: stand
State: (20, 10, True), Action: stand
State: (21, 1, False), Action: stand
State: (21, 1, True), Action: stand
State: (21, 2, False), Action: stand
State: (21, 2, True), Action: stand
State: (21, 3, False), Action: stand
State: (21, 3, True), Action: stand
State: (21, 4, False), Action: stand
State: (21, 4, True), Action: stand
State: (21, 5, False), Action: stand
State: (21, 5, True), Action: stand
State: (21, 6, False), Action: stand
State: (21, 6, True), Action: stand
State: (21, 7, False), Action: stand
State: (21, 7, True), Action: stand
State: (21, 8, False), Action: stand
State: (21, 8, True), Action: stand
State: (21, 9, False), Action: stand
State: (21, 9, True), Action: stand
State: (21, 10, False), Action: stand
State: (21, 10, True), Action: stand
```

In this notebook, I have implemented and analyzed various Blackjack playing agents using Reinforcement Learning techniques. The code includes detailed evaluations, hyperparameter tuning, comparative performance analysis, and visualizations of Q-values, cumulative rewards, and exploration vs. exploitation actions.

The final results demonstrate the effectiveness of different strategies and highlight the impact of rule variations on the performance of the agents.