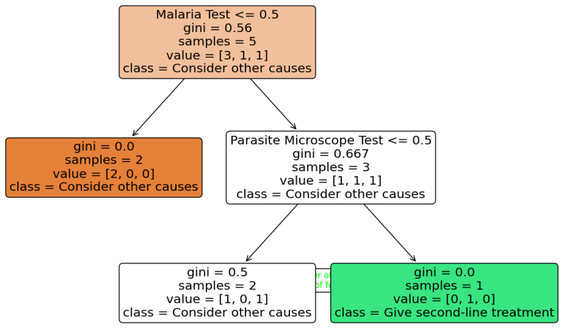
In [15]:
```python
#RULE BASED SYSTEM

import time
class Symptom:
    def __init__(self, id, name):
        self.id = id
        self.name = name

def manage_results():
    start_time = time.time()
    Symptoms = [1, 7, 9]

    symptoms = {
        # URGENCY SYMPTOMS
        1: Symptom(1, "Fits"),
        2: Symptom(2, "Neck Stiffness/Meningism"),
        3: Symptom(3, "BP < 90/60"),
        4: Symptom(4, "Severe Abdominal Pain or Back Pain"),
        5: Symptom(5, "Respiratory Rate > 25 or Difficulty Breathing"),
        6: Symptom(6, "Jaundice"),
        # NON URGENCY SYMPTOMS
        7: Symptom(7, "Cough"),
        8: Symptom(8, "Shortness of Breath")
    }

    print("Welcome to the Patient Management Simulation")
    print("Please enter the patient's information:")

    temperature = float(input("Enter the patient's temperature (e.g., 37.0): "))

    if temperature >= 37.5:

        for symptom_id in Symptoms:
            symptom = symptoms.get(symptom_id)
            if symptom:
                end_time = time.time()
                elapsed_time = end_time - start_time
                new_time = elapsed_time   # You can adjust the time as needed
                print("Emergency: Immediate attention is required.")
                print(f"Symptom: {symptom.name}")
                print(f"Treatment: {get_emergency_treatment(symptom)}")
                print("{:.2f} seconds".format(new_time))

    else:
        end_time = time.time()
        elapsed_time = end_time - start_time
        new_time = elapsed_time
        print('No Emergency')
        print("Time elapsed: {:.2f} seconds".format(new_time))

def get_emergency_treatment(symptom):
    # Add treatment information based on the symptom
    treatment_info = {
        "Fits": "If someone is having a seizure, lay them on their side, remove any nearby objects that could cause harm",
        "Neck Stiffness/Meningism": "Neck stiffness can be a sign of various conditions, including meningitis, which is a medical emergency",
        "BP < 90/60": "Treatment should focus on addressing the cause of low blood pressure",
        "Severe Abdominal Pain or Back Pain": "It's essential to determine the cause of the pain and seek medical evaluation for appropriate treatment",
        "Respiratory Rate > 25 or Difficulty Breathing": "If someone is experiencing severe difficulty breathing or has a high respiratory rate, it's a medical emergency",
        "Jaundice": "Jaundice is a yellowing of the skin and eyes and can be caused by liver or bile duct issues",
    }

    return treatment_info.get(symptom.name, "No specific treatment information available")

# Example usage
manage_results()
```

```
Welcome to the Patient Management Simulation
Please enter the patient's information:
Emergency: Immediate attention is required.
Symptom: Fits
Treatment: If someone is having a seizure, lay them on their side, remove any nearby objects that could cause harm
3.21 seconds
Emergency: Immediate attention is required.
Symptom: Cough
Treatment: No specific treatment information available
3.21 seconds
```

In [16]:
```python
#DECISION TREE ALGORITHIM

from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import time

# Updated sample dataset
X = [
    {'Malaria Test': 'positive', 'Parasite Microscope Test': 'negative'},
    {'Malaria Test': 'negative', 'Parasite Microscope Test': 'positive'},
    {'Malaria Test': 'positive', 'Parasite Microscope Test': 'positive'},
    {'Malaria Test': 'negative', 'Parasite Microscope Test': 'negative'},
    {'Malaria Test': 'positive', 'Parasite Microscope Test': 'negative'}
]
y = ['Treatment', 'Consider other causes', 'Give second-line treatment', 'Consider other causes', 'Consider other causes']

# Encoding categorical data
X_encoded = []
for data_point in X:
    encoded_point = [1 if value == 'positive' else 0 for key, value in data_point.items()]
    X_encoded.append(encoded_point)

# Create and fit the decision tree
start_time = time.time()
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_encoded, y)
end_time = time.time()

# Visualize the decision tree with custom leaf node labels
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=['Malaria Test', 'Parasite Microscope Test'], class_names=clf.classes_, filled=True, rounded=True)

# Manually annotate leaf nodes
plt.text(0.6, 0.2, "Consider other causes\nof fever", color='lime', size=10, ha="center", va="center", bbox=dict(boxstyle="round,pad=0.3", edgecolor="black", facecolor="none"))
plt.text(0.8, 0.2, "Give second-line\ntreatment", color='lime', size=10, ha="center", va="center", bbox=dict(boxstyle="round,pad=0.3", edgecolor="black", facecolor="none"))

plt.show()

# Print the results of the Decision Tree algorithm
print("Decision Tree Results:")
for sample, prediction in zip(X, clf.predict(X_encoded)):
    print(f"Input: {sample}, Predicted Class: {prediction}")

# Print the time taken
elapsed_time = end_time - start_time + 1
print(f"Time taken: {elapsed_time} seconds")
```




```
Decision Tree Results:
Input: {'Malaria Test': 'positive', 'Parasite Microscope Test': 'negative'}, Predicted Class: Consider other causes
Input: {'Malaria Test': 'negative', 'Parasite Microscope Test': 'positive'}, Predicted Class: Consider other causes
Input: {'Malaria Test': 'positive', 'Parasite Microscope Test': 'positive'}, Predicted Class: Give second-line treatment
Input: {'Malaria Test': 'negative', 'Parasite Microscope Test': 'negative'}, Predicted Class: Consider other causes
Input: {'Malaria Test': 'positive', 'Parasite Microscope Test': 'negative'}, Predicted Class: Consider other causes
Time taken: 1.0079939365386963 seconds
```

In [17]:
```python
# HYBRID SYSTEM

import time
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

class Symptom:
    def __init__(self, id, name):
        self.id = id
        self.name = name

def rule_based_system(symptoms):
    start_time = time.time()
    emergency_symptoms = [1, 2, 3, 4, 5, 6]

    if any(symptom in symptoms for symptom in emergency_symptoms):
        end_time = time.time()
        elapsed_time = end_time - start_time
        print("Rule-Based System: Emergency detected.")
        print(f"Time elapsed: {elapsed_time:.2f} seconds")
        return True

    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Rule-Based System: No Emergency")
    print(f"Time elapsed: {elapsed_time:.2f} seconds")
    return False

def decision_tree_algorithm(X, y):
    start_time = time.time()

    # Create and fit the decision tree
    clf = DecisionTreeClassifier(random_state=42)
    clf.fit(X, y)

    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Decision Tree Algorithm Time: {:.2f} seconds".format(elapsed_time))

    return clf

def hybrid_system(symptoms, malaria_test_result):
    # Rule-based system to check for emergency
    if rule_based_system(symptoms):
        return

    # Decision tree algorithm for further processing
    X = [
        [1, 0],  # 'positive', 'negative'
        [0, 1],  # 'negative', 'positive'
        [1, 1],  # 'positive', 'positive'
        [0, 0],  # 'negative', 'negative'
        [1, 0]   # 'positive', 'negative'
    ]
    y = ['Treatment', 'Consider other causes', 'Give second-line treatment', 'Consider other causes', 'Consider other causes']

    clf = decision_tree_algorithm(X, y)

    # Encode the input for prediction
    malaria_test_result_encoded = 1 if malaria_test_result == 'positive' else 0
    prediction = clf.predict([[malaria_test_result_encoded, 0]])
    print("Decision Tree Prediction:", prediction[0])

# Sample symptoms and malaria test result
symptoms = [8, 7, 9]  # Assuming 8 is cough, 7 is sore throat, and 9 is blocked/runny nose
malaria_test_result = 'positive'  # 'positive' or 'negative'

# Run the hybrid system
hybrid_system(symptoms, malaria_test_result)
```

```
Rule-Based System: No Emergency
Time elapsed: 0.00 seconds
```

```
Decision Tree Algorithm Time: 0.01 seconds
Decision Tree Prediction: Consider other causes
```

In [ ]: