Aim : To implement Monte Carlo Policy Control to optimally assign incoming customer service calls to available representatives in order to minimize the average call handling time in a call center.

Algorithm: Monte Carlo Policy Control (ε-greedy)

Initialize

- Initialize the action-value function Q(s,a) arbitrarily.
- Initialize returns sum and count for all state-action pairs.
- Set exploration rate ϵ

For each episode

- Randomly generate a call type (state).
- Select a representative (action) using an ε-greedy policy based on Q(s,a).
- Simulate the handling time for the selected representative.
- Compute reward as negative handling time.
- Update returns and compute the mean return for Q(s,a).

Policy Improvement

- Continuously update the policy to select the action with the highest expected reward.

Termination

- After sufficient episodes, extract the learned optimal assignment policy.

code:

```
import numpy as np

import random

from collections import defaultdict


# -----------------------------

# Environment Setup

# -----------------------------
```

```
NUM_CALL_TYPES = 3
NUM_REPS = 3

# Mean handling times (minutes)
# Rows = call types, Columns = representatives
MEAN_HANDLING_TIMES = np.array([
    [4.0, 6.0, 5.0],   # Billing
    [7.0, 4.0, 6.0],   # Technical
    [5.0, 5.0, 4.0]    # General
])

def simulate_call(call_type, rep):
    """Simulate call handling time"""
    mean_time = MEAN_HANDLING_TIMES[call_type, rep]
    return max(1.0, np.random.normal(mean_time, 0.5))

# -----------------------------
# Monte Carlo Policy Control
# -----------------------------

Q = defaultdict(lambda: np.zeros(NUM_REPS))
returns_sum = defaultdict(lambda: np.zeros(NUM_REPS))
returns_count = defaultdict(lambda: np.zeros(NUM_REPS))

epsilon = 0.1
```

```
episodes = 20000

def epsilon_greedy(state):
    if random.random() < epsilon:
        return random.randint(0, NUM_REPS - 1)
    return np.argmax(Q[state])

# -----------------------------
# Training
# -----------------------------

for _ in range(episodes):
    state = random.randint(0, NUM_CALL_TYPES - 1)
    action = epsilon_greedy(state)

    handling_time = simulate_call(state, action)
    reward = -handling_time

    returns_sum[state][action] += reward
    returns_count[state][action] += 1
    Q[state][action] = returns_sum[state][action] / returns_count[state][action]

# -----------------------------
# Display Result
# -----------------------------
```

```
call_types = ["Billing", "Technical", "General"]
print("Optimal Call Assignment Policy:\n")

for s in range(NUM_CALL_TYPES):
    best_rep = np.argmax(Q[s])
    print(f"{call_types[s]} Call → Assign to Representative {best_rep}")
```

Output:

Optimal Call Assignment Policy:

Billing Call → Assign to Representative 0

Technical Call → Assign to Representative 1

General Call → Assign to Representative 2

Result:

Call Type	Optimal Representative
Billing	Representative 0
Technical	Representative 1
General	Representative 2