In [1]:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import random
```

```
C:\Python310\lib\site-packages\torch\utils\_pytree.py:185: FutureWarning: op
tree is installed but the version is too old to support PyTorch Dynamo in C+
+ pytree. C++ pytree support is disabled. Please consider upgrading optree u
sing `python3 -m pip install --upgrade 'optree>=0.13.0'`.
  warnings.warn(
```

In [2]:

```
# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
```

Out[2]:

```
<torch._C.Generator at 0x274fd813210>
```

In [3]:

```
class MultiAgentEnvironment:
    def __init__(self):
        self.state_dim = 6  # [ball_x, ball_y, keeper_x, keeper_y, taker_x, ta
        self.action_space = [
            np.array([0.0, 0.1]),   # up
            np.array([0.0, -0.1]),  # down
            np.array([-0.1, 0.0]),  # left
            np.array([0.1, 0.0]),   # right
            np.array([0.0, 0.0])    # stay
        ]
        self.action_dim = len(self.action_space)
        self.max_steps = 50
        self.reset()

    def reset(self):
        self.keeper_pos = np.array([0.0, 0.0])
        self.taker_pos = np.array([random.uniform(-1, 1), random.uniform(-1, 1
        self.ball_pos = np.array([random.uniform(-1, 1), random.uniform(-1, 1)
        self.steps = 0
        return self._get_state()

    def _get_state(self):
        return np.concatenate([self.ball_pos, self.keeper_pos, self.taker_pos]

    def step(self, keeper_action_idx, taker_action_idx):
        keeper_action = self.action_space[keeper_action_idx]
        taker_action = self.action_space[taker_action_idx]

        self.keeper_pos += keeper_action
        self.taker_pos += taker_action

        # Ball moves toward the taker
        direction = self.taker_pos - self.ball_pos
        if np.linalg.norm(direction) > 0:
            self.ball_pos += 0.1 * direction / np.linalg.norm(direction)

        # Rewards: Keeper tries to reach ball, Taker tries to steal it
        keeper_reward = -np.linalg.norm(self.keeper_pos - self.ball_pos)
        taker_reward = -np.linalg.norm(self.taker_pos - self.ball_pos)

        self.steps += 1
        done = self.steps >= self.max_steps
        return self._get_state(), keeper_reward, taker_reward, done
```

In [4]:

```
class QNetwork(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, output_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```

In [7]:

```
class IndependentQLearningAgent:
    def __init__(self, input_dim, action_dim, lr=0.001, gamma=0.99):
        self.q_net = QNetwork(input_dim, action_dim)
        self.optimizer = optim.Adam(self.q_net.parameters(), lr=lr)
        self.gamma = gamma
        self.action_dim = action_dim

    def select_action(self, state, epsilon=0.1):
        if random.random() < epsilon:
            return random.randint(0, self.action_dim - 1)
        with torch.no_grad():
            q_vals = self.q_net(torch.tensor(state, dtype=torch.float32))
            return torch.argmax(q_vals).item()

    def update(self, state, action, reward, next_state):
        state_tensor = torch.tensor(state, dtype=torch.float32)
        next_state_tensor = torch.tensor(next_state, dtype=torch.float32)

        q_vals = self.q_net(state_tensor)
        next_q_vals = self.q_net(next_state_tensor)

        # Convert target to float32 explicitly
        target = reward + self.gamma * torch.max(next_q_vals).item()
        target_tensor = torch.tensor(target, dtype=torch.float32)

        loss = nn.MSELoss()(q_vals[action], target_tensor)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```

In [8]:

```python
if __name__ == "__main__":
    env = MultiAgentEnvironment()
    keeper = IndependentQLearningAgent(env.state_dim, env.action_dim)
    taker = IndependentQLearningAgent(env.state_dim, env.action_dim)

    num_episodes = 1000

    for ep in range(num_episodes):
        state = env.reset()
        total_keeper_reward = 0
        total_taker_reward = 0

        done = False
        while not done:
            keeper_action = keeper.select_action(state)
            taker_action = taker.select_action(state)

            next_state, kr, tr, done = env.step(keeper_action, taker_action)

            keeper.update(state, keeper_action, kr, next_state)
            taker.update(state, taker_action, tr, next_state)

            total_keeper_reward += kr
            total_taker_reward += tr
            state = next_state

        print(f"Episode {ep + 1}/{num_episodes}, Keeper Reward: {total_keeper_
```

```
Episode 981/1000, Keeper Reward: -145.70, Taker Reward: -2.99
Episode 982/1000, Keeper Reward: -146.37, Taker Reward: -23.75
Episode 983/1000, Keeper Reward: -95.52, Taker Reward: -21.73
Episode 984/1000, Keeper Reward: -139.91, Taker Reward: -0.27
Episode 985/1000, Keeper Reward: -137.55, Taker Reward: -1.14
Episode 986/1000, Keeper Reward: -169.37, Taker Reward: -17.96
Episode 987/1000, Keeper Reward: -112.29, Taker Reward: -6.09
Episode 988/1000, Keeper Reward: -105.72, Taker Reward: -5.95
Episode 989/1000, Keeper Reward: -112.49, Taker Reward: -1.93
Episode 990/1000, Keeper Reward: -132.00, Taker Reward: -24.12
Episode 991/1000, Keeper Reward: -115.13, Taker Reward: -34.57
Episode 992/1000, Keeper Reward: -99.75, Taker Reward: -3.78
Episode 993/1000, Keeper Reward: -135.40, Taker Reward: -4.23
Episode 994/1000, Keeper Reward: -114.02, Taker Reward: -2.52
Episode 995/1000, Keeper Reward: -103.46, Taker Reward: -19.43
Episode 996/1000, Keeper Reward: -173.40, Taker Reward: -13.99
Episode 997/1000, Keeper Reward: -88.63, Taker Reward: -54.55
Episode 998/1000, Keeper Reward: -100.33, Taker Reward: -22.46
Episode 999/1000, Keeper Reward: -112.30, Taker Reward: -34.53
Episode 1000/1000, Keeper Reward: -111.60, Taker Reward: -38.59
```

In [ ]: