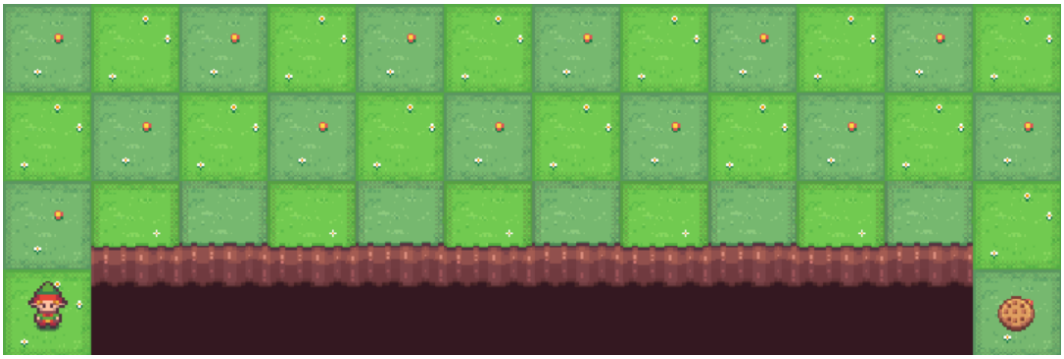
فرآیند تصمیم مارکوف (MDP)



سیدمحمدحسین هاشمی ۴۰۲۲۳۶۳۱۴۳
سیدحسین حسینی ۴۰۱۲۳۶۳۲۳۸

آبان ۱۴۰۲

فهرست مطالب

۱ لیست صخره ها

cliffwalking_MDP.py

```
traps = [float(i[0] * 12 + i[1]) for i in env.cliff_positions]
```

شماره خانه های صخره درون لیست traps ذخیره می‌شود.

۲ آماده سازی دیکشنری states

cliffwalking_MDP.py

```
states = dict()
for i in range(0, 4):
    for j in range(0, 12):
        if i * 12 + j in traps:
            states[i * 12 + j] = -100
        elif i == 3 and j == 11:
            states[i * 12 + j] = 1000
        else:
            states[i * 12 + j] = {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0}
```

از دیکسشنری states برای محاسبات استفاده می‌کنیم. در اینجا مقادیر اولیه آن را تعریف کرده‌ایم.

۳ تابع محاسبه امتیاز خانه

cliffwalking_MDP.py

```
def calculateScore(inp):
    global states
    return max(states[inp].values()) if isinstance(states[inp], type(dict())) else states[inp]
```

این تابع ارزش خانه فراخوانی شده را برمی‌گرداند. بدین صورت که در صورت بودن هدف و یا صخره مستقیما امتیاز آن برگشت داده می‌شود و در غیر اینصورت مقدار بیشترین عمل برگشت داده می‌شود.

۴ تابع به روز کردن ارزش عمل

cliffwalking_MDP.py

```
def changeScore(kp, ks, v1, v2, v3):
    global states
    score = (v1 + v2 + v3) / 3
    change = score - states[kp][ks]
    states[kp][ks] = score
    return abs(change)
```

از این تابع برای بروز مقدار ارزش هر عمل استفاده می‌شود و مقدار بازگشت داده شده مقدار تغییرات آن است.

۵ تابع بروز کردن مقادیر states

cliffwalking_MDP.py

```python
def updateState():
    global states
    totalChanges = 0
    for Key, Val in states.items():
        if isinstance(Val, type(dict())):
            row = Key // 12
            col = Key % 12
            up = row * 12 + col if row == 0 else (row - 1) * 12 + col
            right = row * 12 + col if col == 11 else row * 12 + col + 1
            down = row * 12 + col if row == 3 else (row + 1) * 12 + col
            left = row * 12 + col if col == 0 else row * 12 + col - 1
            for Ks, Vs in Val.items():
                upScore = calculateScore(up)
                rightScore = calculateScore(right)
                leftScore = calculateScore(left)
                downScore = calculateScore(down)

                if Ks == 0:
                    totalChanges += changeScore(Key, Ks, upScore, rightScore, leftScore)
                elif Ks == 1:
                    totalChanges += changeScore(Key, Ks, upScore, rightScore, downScore)
                elif Ks == 2:
                    totalChanges += changeScore(Key, Ks, leftScore, rightScore, downScore)
                elif Ks == 3:
                    totalChanges += changeScore(Key, Ks, upScore, rightScore, downScore)
    return totalChanges
```

این تابع اصلی برای بروز کردن مقادیر states است.

بدین صورت که با هر بار اجرا بصورت مستقل عمل می‌کند و هر احرای آن هر خانه و عمل‌های آن را ارزش‌گذاری و مقدار کل تغییرات را برمی‌گرداند.

پس از پیمایش states درصورتی که خانه صخره و یا هدف نبود، موقعیت بعدی آن را از هر سمت به دست می‌آوریم و سپس امتیاز اعمال را با استفاده از توابع بالا بروز رسانی می‌کنیم.

۶ حلقه مارکوف

cliffwalking_MDP.py

```
n = 1000
changes = updateState()

while n > 0 and changes > 0.005:
    changes = updateState()
    n -= 1

policies = dict()
for Kp, Vp in states.items():
    if isinstance(Vp, type(dict())):
        Kmax = 1
        VMax = Vp[1]
        for K, V in Vp.items():
            if V > VMax:
                Kmax = K
                VMax = V
        policies[Kp] = Kmax

policies[47] = 1
```

این حلقه همان حلقه معروف فرایند مارکوف است که در صورت کم نشدن مقدار تغییرات هزار بار تابع updateState را فراخوانی و مقدار states را بروز می‌کند.

بعد از آن policies را با استفاده از states ایجاد کرده‌ایم، بدین صورت که با ارزش ترین عمل انتخواب می‌شود.

در نهایت برای اینکه خانه ۴۷ خانه هدف است حساب نشده که برای رفع باگ مقدار ۱ (حرکت به راست) را برای آن قرارداده‌ایم.

۷ لیست صخره ها

cliffwalking_MDP.py

```python
# Define the maximum number of iterations
max_iter_number = 1000
next_state = 36
winRate = 0
sumRewards = 0
for __ in range(max_iter_number):
    # TODO: Implement the agent policy here
    # Note: .sample() is used to sample random Action from the environment's Action space

    # Choose an Action (Replace this random Action with your agent's policy)
    # Action = env.action_space.sample()
    action = policies[next_state]

    # Perform the Action and receive feedback from the environment
    next_state, reward, done, truncated, info = env.step(action)

    sumRewards += reward

    if done:
        winRate += 1
        sumRewards += 1000

    if done or truncated:
        observation, info = env.reset()

# Close the environment
env.close()
```

با قرار دادن policies در کد کار به اتمام می‌رسد.