HW1 Report 2020203090 한용옥

본 강화학습의 모델은 격자세계에서의 이동을 다루고 있다 모델의 MDP 5요소는 아래와 같다

S	격자점의 좌표 (s_y, s_x)
A	상하좌우, 대각선 총 8방향 이동
$p(s', r \mid s, a)$	결정적 전이확률
R	결정적 보상
γ	상수로 조정 가능

프로그램의 `step` 함수는 MDP 전이확률을 구현한다 Deterministic Transition Dynamics를 가정하므로 아래의 수식이 성립한다

$$P(S_{t+1} = s, R_{t+1} = r | S_t = s_0, A_t = a_0) = \begin{cases} 1 & \text{if } s = s_1, r = r_1 \\ 0 & \text{otherwise} \end{cases}$$

`step` 함수는 상태(= 좌표), 행동(= 이동 벡터)를 받아 다음 상태와 보상을 반환한다
결정적이고 둘 다 좌표이므로 다음 상태는 단순히 둘을 더하면 된다 따라서 아래와 같이 구현한다

```
def step(state, action):
    if state in terminal_states:
        return state, 0.0
    dx, dy = action

    # TODO ============================================
    next_state = (state[0] + action[0], state[1] + action[1])
    # ==================================================

    if not is_bounded(next_state):
        return state, WALL_BUMP_COST
    distance = math.sqrt(dx**2 + dy**2)
    return next_state, STEP_COST * distance
```

본 프로그램에선 가치반복 알고리즘을 사용하여 최적 정책을 찾는다 아래는 스켈레톤 코드의 의사코드다

```
#psuedo-code
def value_iteration(grid_size, gamma, theta, max_iterations):
    V = 0
    for k in range(max_iterations):
        delta = 0
        for grid_point in grid_size:
            if terminal:
                continue
            values = np.zeros(num_actions)
            for a in action:
                # TODO ============================================
                values[idx] =
                # ================================================
            # TODO ============================================
            V[i, j] =
            delta =
            # ================================================
            best_action = np.argmax(values)
        if delta < theta:
            break
    return
```

따라서 채워야 할 부분은 가치반복 알고리즘의 가치를 최대로 반복 계산하는 부분이다 구현 접근은 다음과 같다
상태는 격자점의 좌표이므로 벨만 최적 재귀식 수식은 아래와 같다

$$V((s_y, s_x)) = max_a \sum_{(s'_y, s'_x), r} p((s'_y, s'_x), r \mid (s_y, s_x), a)(r + \gamma V((s'_y, s'_x)))$$

결정적 전이모델이므로 특정 상태에서 특정 행동을 할 때 한 가지 상태로의 전이확률만 1이고 나머지는 0이다 따라서 아래와 같이 줄일 수 있다

$$V((s_y, s_x)) = max_a(r + \gamma V((s'_y, s'_x)))$$

이 수식으로 부터 `values[idx] =` 는 max_a 에 대한 후보 $r + \gamma V((s'_y, s'_x))$를 구하는 부분
`V[i, j] =` 는 구해진 후보로부터 최대값을 뽑아 대입하는 부분임을 알 수 있다
Δ 는 수렴 오차를 판별하기 위한 변수이며 $\Delta = max(\Delta, |V_{old}(s) - V_{new}(s)|)$ 로 구해진다
모두 종합해 TODO 부분은 아래와 같이 구현 하였다

```
    values[idx] = reward + gamma * V[ni, nj]
V[i, j] = np.max(values)
delta = max(delta, np.abs(v[i, j] - V[i, j]))
```

추가 간단 실험

관찰	주어진 코드의 `WALL_BUMP_COST` 는 `-1.0`로 보드 내에서의 정상적 상하좌우이동과 같다 대각선 이동보다는 더 큰 값이다
가설	`WALL_BUMP_COST`가 정상적 이동의 보상과 같다면 밖으로 나가는 이동의 빈도가 정상과 같아져 수렴속도가 느려질 것
설계	$\gamma = 0.9, \theta = 10^{-3}$, `STEP_COST = -1.0` 고정 `WALL_BUMP_COST = -1, -10, -100, -100000`을 비교
결과	순서대로 $7, 6, 6, 6$번 만에 수렴
해석	정상이동 보다 클 때 반복횟수가 줄어들었으나 비중있게 줄어들진 않았다

관찰	주어진 코드의 γ 는 `0.9`로 비교적 먼 미래까지 관찰한다 상하좌우이동의 보상은 -1이지만 대각선이동은 $-\sqrt{2}$다 대각선이동으로 두 칸 이동하는 대신 좀 더 나쁜 보상을 받는다
가설	γ가 작아질수록 직전의 미래만 참고해 두 칸 이동할 수 있지만 나쁜 보상을 받는 대각선 이동을 하지 않을 것
설계	$\theta = 10^{-3}$, `STEP_COST = -1.0`, `WALL_BUMP_COST = -100` 고정 $\gamma = 0, 0.25, 0.5, 0.75, 1$을 비교 수렴 상태에서 대각선 이동의 횟수를 세어 비교
결과	순서대로 (총 대각선 이동 개수, 수렴에 이르기까지의 횟수) (0, 2), (0, 6), (11, 7), (18, 7), (25, 6)
해석	γ가 클수록 더 먼 미래를 보기 때문에 보상은 약간 작지만 더 크게 이동을 할 수 있는 대각선 이동을 더 많이 사용한다