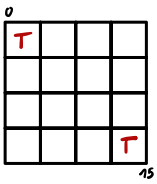
1. DP - Policy evaluation

Fixed π → state-value function V^π for MDP:



A: ↑↓←→
step on T: R = -1
T: R = 0

Stop: $\Delta v < \phi$

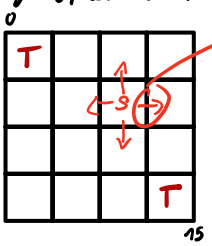
def policy_eval(π, env, γ=1, φ=0.00001):

$$V(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$$

for each s: $v = \sum_a \pi(a|s) \sum P(s'|s,a)[r + \gamma V(s')]$

2. Policy Iteration

Same grid, actions...:



we need this basis for our improvements

we use the V(s) from our eval

Eg. V(s) = -17 → compute Q(s,a)

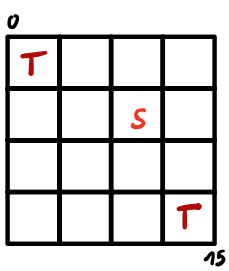
→ we update policy for this s to always go right

- Now run policy_eval again but with our improved policy
  ⋮
  ~ and get $V^*[s] = -2$

Goal: Given V(s) improve the policy by acting greedily

For each state s: choose action that maximizes expected return

Same as 1 & 2:



3 Value Iteration

- updates the value of each state by considering the best possible action → we extract optimal policy from it.

- after convergence the optimal policy is extracted
- requires full knowledge of env.

4. Gambler's problem

- Find best way to win $100 before going broke
- V[s] shows chance of reaching $100 from capital s
- using value iteration for best optimal policy
- optimal policy: often bet big (depends on p_h)

```
def one_step_lookahead:
    for each state in capital:
        expected = ph * (reward_win + V[win_cap]) + (1-ph) * (reward_lose + V[lose_cap])
        append to actions[]
def: val_it_gambler:
    init...
    until delta < theta:
        V[s] = max(one_step_lookahead)

for states 1-99:
    action_vals = one_step_lookahead
    best_action = max(...) + 1
    policy[s] = ...
```