MDPs - Algorithms	start time:

Before you start, share this document with your team member(s) and then complete the form below to assign the role of speaker.

Team Role	Team Member
Speaker: shares your team's ideas with the class.	Makenna, Arogya, Jesse

A. Review of Grid World	start time:

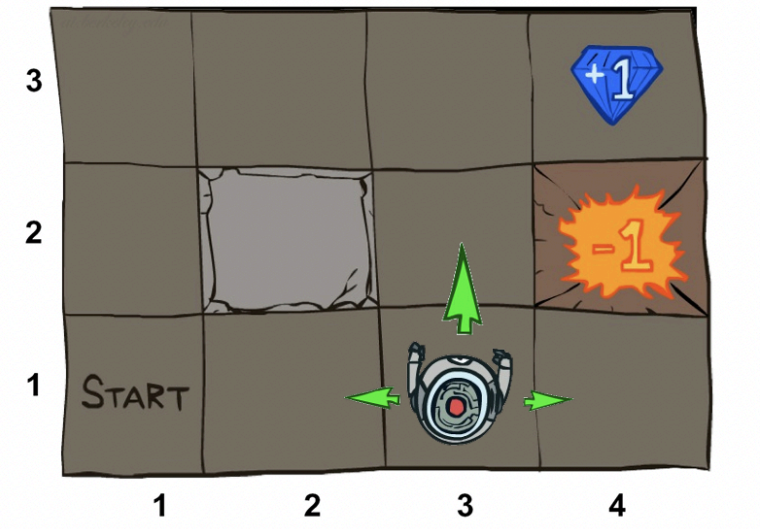
Recall the GridWorld we have examined, where there are only two terminal states: state (3, 4) and state (2, 4), with rewards 1 and -1. All other transitions between non-terminal states have a -0.04 reward (penalty).

The discount rate gamma is 0.5.

In the following grid, for each state, there are five possible actions (N, S, E, W, and exit). Each state has 80% chance to move following the planned action and 10% chance to move following the perpendicular direction against the planned action. If it hits a wall, it does not move.

For an example action N on (3,3), there are three non-zero transition probabilities when you start at state (3,3):

T((3,3), N, (3,3)) = 0.8
T((3,3), N, (3,2)) = 0.1
T((3,3), N, (3,4)) = 0.1



1. If the bottom left corner state is (1,1),
    a. What are the coordinates of the two terminal states?

        (3, 4), (2, 4)

    b. What are the reward values for the two terminal states?

        +1 in (3,4) and -1 in (2,4)

    c. What are the reward values for the non-terminal states?

        -0.04

2. What are the four possible actions for each state?

    N, E, W, S

3. Look at the transition functions in the above model.
    a. What do the values *s, a, and s'* represent?

        S is the current coordinate, A is the action, and S' is the new coordinate after taking the action

    b. Explain why the transition functions *T(s, a, s')* add up to 1.

        They add up to 1 because each function yields a probabilistic value that sums to 100%

4. What is the maximum expected reward at state (2, 3)? Which action leads to the maximum rewards? (Hint: use formula $max_a \ \Sigma_{s'} T(s, a, s') * R(s, a, s')$, that is finding the action that maximizes the expected rewards.)

    Planned: E
    80% chance: T((2, 3), E, (2, 4)) is -1 = 80/100 * -1 = -0.8
    10% chance: T((2, 3), N, (3, 3)) is -0.04 = 10/100 * -0.04 = -0.004
    10% chance: T((2, 3), S, (1, 3)) is -0.04 = 10/100 * -0.04 = -0.004

    Sum = -0.808

    Planned: W
    80% chance: T((2, 3), W, (2,3)) is -0.04 = 80/100 * -0.04 =-0.032
    10% chance: T((2, 3), N, (3,3)) is -0.04 = 10/100 * -0.04 = -0.004
    10% chance: T((2, 3), S, (1,3)) is -0.04 = 10/100 * -0.04 = -0.004

    Sum = -0.036

    Planned: N
    80% chance: T((2, 3), N, (3, 3)) is -0.04 = 80/100 * -0.04 = -0.032
    10% chance: T((2, 3), W, (2, 3)) is -0.04 = 10/100 * -0.04 = -0.004
    10% chance: T((2, 3), E, (2, 4)) is -1 = 10/100 * -1 = -0.1

    Sum = -0.136

    Planned: S
    80% chance: T((2, 3), S, (1, 3)) is -0.04 = 80/100 * -0.04 = -0.032
    10% chance: T((2, 3), E, (2, 4)) is -1 = 10/100 * -1 = -0.1
    10% chance: T((2, 3), W, (2, 3)) is -0.04 = 10/100 * -0.04 = -0.004

    Sum = -0.136

B. The Bellman Equation	start time:

In order to talk about the Bellman equation for MDPs, we must first introduce two new mathematical quantities:

- The optimal value of a state s, $U^*(s)$ – the optimal value of s is the expected value of the utility an optimally-behaving agent that starts in s will receive, over the rest of the agent's lifetime. Note that frequently in the literature the same quantity is denoted with $V^*(s)$.
- The optimal value of a Q-state (s,a), $Q^*(s,a)$ - the optimal value of (s,a) is the expected value of the utility an agent receives after starting in s, taking a, and acting optimally henceforth.

Using these two new quantities and the other MDP quantities discussed earlier, the Bellman equation is defined as follows:

$$U^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$$

Before we begin interpreting what this means, let's also define the equation for the optimal value of a Q-state (more commonly known as an optimal **Q-value**):

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$$

Note that this second definition allows us to reexpress the Bellman equation as

$$U^*(s) = \max_a Q^*(s,a)$$

5. What is state s and q-state (s,a)? Write the answer in your own words.

   State s is the current state
   q is pair of state and action

6. What is the state value U(s) and q-value Q(s,a) following a policy π? Write the answer in your own words.

   State value- expected value from the next step not the optimal value
   q-value- expected value not optimal value

7. What is the optimal state value U*(s) and optimal q-value Q*(s,a)? Write the answer in your own words.

   U*(s) - the best next action given our policy. The maximum value we can receive from our possible next steps.

   The Q value - max reward we can obtain based on all given moves following the policy.

8. What is the relationship between optimal state value and optimal q-value?

   An optimal q-value is the optimal path of actions (highest possible reward) after action a. The optimal state value is the best value of all available optimal q-values.

9. Explain the meaning of this term: R(s,a,s′)+γU*(s′).

   The reward value of a transition + gamma * value of next state.

   The best reward we can get from this action

   - The new position s' might not be the optimal place to be landed on

10. Explain the meaning of this term: T(s,a,s') * (R(s,a,s′)+γU*(s′)).

    Multiplying the best reward by the probabilistic value for the transition.

    - The new position s' will be the optimal since the probability is taken into account

11. Based on your understanding in question 7, explain wh

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma U^*(s')\right)$$

    This is the sum of all transition rewards.

12. Based on your understanding in question 8, explain why

$$U^*(s) = \max_a Q^*(s,a)$$

    We are choosing the highest Q* value as our U* for s.

    It gives us the best next action based on all of the Q* values and we are choosing 1 by looking at all of the possible next actions.

13. Combining equations in 11 and 12, we have

$$U^*(s) = \max_a \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma U^*(s')\right)$$

    This equation is the Bellman equation. We could use the right side of the equation to calculate the left side value U*(s). However, do you think U*(s) on the left side could be one of U*(s') on the right side?

    It could be the same. If the next best action is the previous action where we came from then we are stuck in a forever loop.

14. If your answer to 13 is yes, can we update a new value of U*(s) without knowing the current value of U*(s)?

    Yes, we do need to have an initial value of U*(s), otherwise we reach a contradiction.

15. Suppose the current optimal values are U*((1,1)) = 1, U*((2,1))=0.5, U*((1,2))=0.3, please calculate the q-value of state (1,1) and action North. Assuming any moving from a non-terminal state costs -0.04, and the discounting rate is 0.9. Calculate the q((1,1), N).

    q((1,1), N) = 0.437

    T((1,1), N, (2,1)) = 0.8 * (-0.04 + (0.9 * 0.5)) = 0.328
    T((1,1), W, (1,1)) = 0.1 * (-0.04 + (0.9 * 1)) = 0.086
    T((1,1), E, (1, 2)) = 0.1 * (-0.04 + (0.9 * 0.3)) = 0.023

    T(s,a,s') * (R(s,a,s′)+γU*(s′))

    0.328 + 0.086 + 0.023 = 0.437

16. Repeat the calculation to get the q-values of state (1,1) and other three actions.

    q((1,1), S) = .797

    T((1,1), S, (1,1)) = 0.8 * (-0.04 + (0.9 * 1)) = 0.688
    T((1,1), W, (1,1)) = 0.1 * (-0.04 + (0.9 * 1)) = 0.086
    T((1,1), E, (1, 2)) = 0.1 * (-0.04 + (0.9 * 0.3)) = 0.023

    T(s,a,s') * (R(s,a,s′)+γU*(s′))

    ---

    q((1,1), E) = .248

    T((1,1), E, (1,2)) = 0.8 * (-0.04 + (0.9 * 0.3)) = 0.184
    T((1,1), N, (2,1)) = 0.1 * (-0.04 + (0.9 * 0.5)) = 0.041
    T((1,1), S, (1, 1)) = 0.1 * (-0.04 + (0.9 * 1)) = 0.086

$$q((1,1), W) = \sum_{s'} T(s,a,s') * (R(s,a,s')+\gamma U^*(s'))$$

$$q((1,1), W) = .815$$

$$T((1,1), W, (1,1)) = 0.8 * (-0.04 + (0.9 * 1)) = 0.688$$
$$T((1,1), N, (2,1)) = 0.1 * (-0.04 + (0.9 * 0.5)) = 0.041$$
$$T((1,1), S, (1, 1)) = 0.1 * (-0.04 + (0.9 * 1)) = 0.086$$

$$U^*((1,1)) = \max_{a} \sum_{s'} T(s,a,s') * (R(s,a,s')+\gamma U^*(s'))$$

17. Based on the 15 and 16, what is the new optimal value U*((1,1)) now?

    U*((1, 1)) = 0.815 (W)

18. How to update the value of U*((1,1)) if we do not know current value U* on the board?

    We cannot update U* if we do not know the current value, unless we are provided an initial value.

C. Value Iteration	start time:

Now that we have a framework to test for optimality of the values of states in a MDP, the natural follow-up question to ask is how to actually compute these optimal values from empty?

The time-limited value for a state s with a time-limit of k timesteps is denoted U_k(s), and represents the maximum expected utility attainable from s given that the Markov decision process under consideration terminates in k timesteps.

Value iteration is a **dynamic programming algorithm** that uses an iteratively longer time limit to compute time-limited values until convergence (that is, until the U values are the same for each state as they were in the past iteration: $\forall s, U_{k+1}(s) = U_k(s)$). It operates as follows:

1. $\forall s \in S$, initialize $U_0(s) = 0$. This should be intuitive, since setting a time limit of 0 timesteps means no actions can be taken before termination, and so no rewards can be acquired.
2. Repeat the following update rule until convergence:

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$$

At iteration step *k*=0, all states are initialized to **value** of 0:

0	0	0	**0**
0	**BLANK**	0	**0**
0	0	0	0

At iteration step *k*=1, the exit states *c* and *e* have now exited and are assigned their reward values. All other states remain at 0.

0	0	0	**1.0**
0	**BLANK**	0	**-1.0**
0	0	0	0

For the rest of this mode, we will manually carry out the value iteration formula, which is:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

19. Is this algorithm the same as the dynamical programming we learned in 306?

    Yes, this is similar. We are finding the solution to a small problem and adding to that to find the larger problem.

20. Why someone calls this algorithm as approximate dynamic programming?

    However, we are continuously updating the solution rather than finding the solution once.

21. Look at the first formula in this model. Identify what each variable stands for.

    a. $U_k(s)$
    b. *T(s, a, s')*
    c. *R(s, a, s')*
    d. γ (*gamma)*

22. In this activity, we will only examine five states. To make it easier to refer to each state, we will call these states a, b, c, d and e:

	a	**b**	**c**
	BLANK	**d**	**e**

What are the coordinates of each of these states?

a - (3,2)

b - (3,3)

c - (3,4)

d - (2,3)

e - (2,4)

23. Explain why the states c and e have 1.0 and -1.0 values, not 0.9 and -0.9 values if γ (*gamma)* is 0.9.

    These are terminal states, the U* is the terminal function

24. The value will be set to the maximum of the q-values. For each state, we need to calculate the q-value for each action.

    So, for state *b* and action *a* = N (for North), the summation produces

    $T(b, N, b)\,[R(b, N, b) + \gamma U_1(b)] + T(b, N, a)\,[R(b, N, a) + \gamma U_1(a)] + T(b, N, c)\,[R(b, N, c) + \gamma U_1(c)]$

    Assuming γ = 0.9, calculate this q-value (the sum) for state *b*, action a=N, iterative step k=2.

    $0.8 * [R(b, N, b) + 0.9 * 0] + 0.1 * [R(b, N, a) + 0.9 * 0] + 0.1 * [R(b, N, c) + 0.9 * 1]$

    0.09 - 0.04 = 0.05

25. Using the same approach, calculate the *q(s, a)* values (at iteration step *k*=2) for the three other action possibilities (S, E, and W).

    South

    $0.8 * [R(b, S, d) + 0.9 * 0] + 0.1 * [R(b, W, a) + 0.9 * 0] + 0.1 * [R(b, E, c) + 0.9 * 1]$

    0.8 * -0.04 = -0.032
    0.1 * -0.04 = -0.004
    0.1 * -0.04 + 0.9 = 0.896

    0.896 - 0.04 - 0.032 = 0.824

    East

    $T(b, N, b)\,[R(b, N, b) + \gamma U_1(b)] + T(b, S, d)\,[R(b, S, d) + \gamma U_1(d)] + T(b, E, c)\,[R(b, E, c) + \gamma U_1(c)]$

    $0.1 * (-0.04 + 0.9 * 0) + 0.1 * (-0.04 + 0.9 * 0) + 0.8 * (-0.04 + 0.9 * 1) = 0.68$

    West

    $0.8(-0.04 + 0.9 * 0) + 0.1(-0.04 + 0.9 * 0) + 0.1(-0.04 + 0.9 * 0) = -0.04$

26. Which action led to the highest value (max over possible actions *a*)? This is the action that the policy should use, and this highest value then becomes the $U_{k+1}(s)$ for this state—i.e., $U_2(b)$.]

    South yielded the highest value.

27. Now calculate $U_2(d)$. What are the actions that should be taken in these two states (in other words, what is their policy)?

28. We can repeat this procedure to calculate the U3 in the next iteration. When should we terminate the iteration?

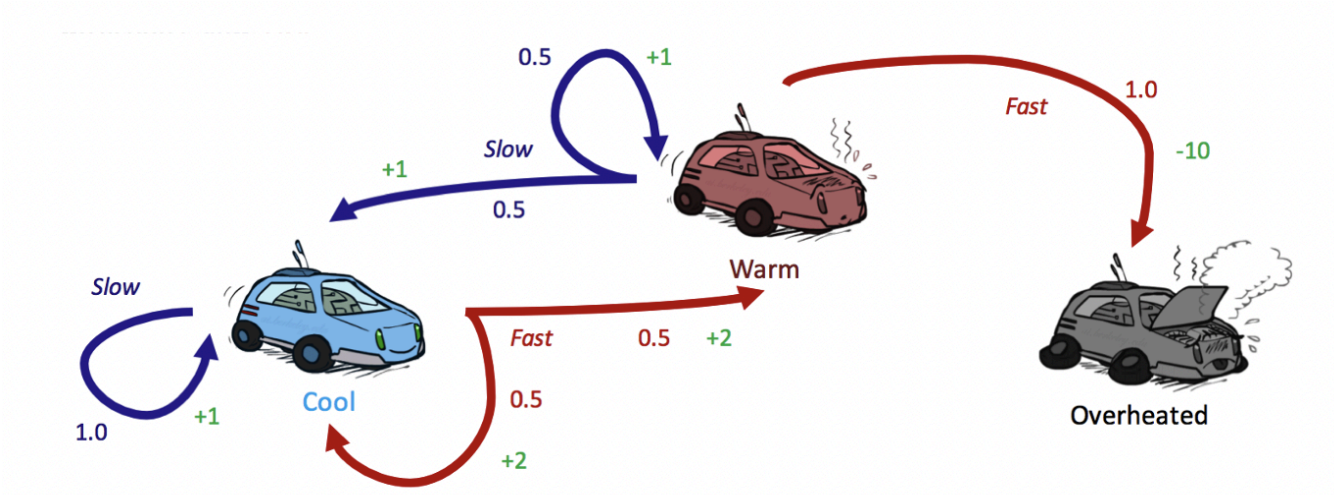
    $U_{max} < \Sigma((1-r)/r)$

    When we need to stop take into consideration the discount rate gamma.

    If gamma is 0 then we only have to look one step ahead. If gamma is 1 we care about every step in the future. In the very far future the reward is seen. We would need more iterations the closer gamma gets to 1.

Exercise

Consider the motivating example of a racecar:



There are three possible states, S = {cool,warm,overheated}, and two possible actions A = {slow, fast}. Just like in a state-space graph, each of the three states is represented by a node, with edges representing actions. Overheated is a terminal state, since once a racecar agent arrives at this state, it can no longer perform any actions for further rewards (it's a sink state in the MDP and has no outgoing edges). Notably, for nondeterministic actions, there are multiple edges representing the same action from the same state with differing successor states. Each edge is annotated not only with the action it represents, but also a transition probability and corresponding reward.

29. List all the transition functions using format T(s, a, s')

    T(cool, fast, warm) = 0.5

    T(cool, slow, cool) = 1.0

    T(cool, fast, cool) = 0.5

    T(warm, fast, overheated) = 1.0

    T(warm, slow, cool) = 0.5

    T(warm, slow, warm) = 0.5

30. List all the reward functions using format R(s, a, s').

    R(cool, fast, warm) = +2

    R(cool, slow, cool) = +1

    R(cool, fast, cool) = +2

    R(warm, fast, overheated) = -10

    R(warm, slow, cool) = +1

    R(warm, slow, warm) = +1

31. We begin value iteration by initialization of all U0(s) = 0

	cool	**warm**	**overheated**
U0	0	0	0

Iterate the values to get U1:

FROM COOL:
T(cool, fast, warm) [R(cool, fast, warm)+ γU_1(*warm*)] + T(cool, slow, cool) [R(cool, slow, cool) + γU_1(cool)] + T(cool, fast, cool) [R(cool, fast, cool) + γU_1(cool)]

0.5 * (2+ 1**0*) + 0.5 (2 + 1**0*) = 2 or 1* (1 + 1**0*) = 1

FROM WARM:
T(warm, fast, overheated) [R(warm, fast, overheated) + γU_1(*overheated*)] + T(warm, slow, cool) [R(warm, slow, cool) + γU_1(cool)] + T(warm, slow, warm) [R(warm, slow, warm) + γU_1(warm)]

= 1* (-10 + 1**0*) + 1* (1 + 1*0) + 1* (1 + 1**0*)
= 1* (-10 + 1**0*) = -10 or 1* (1 + 1*0) = 1 we choose 1

now 1* (1 + 1*0) =1 or 1* (1 + 1**0*) = 1

	cool	**warm**	**overheated**
U0	0	0	0
U1	2	1	0

32. Iterate the values again to get U2:

FROM COOL:
T(cool, fast, warm) [R(cool, fast, warm)+ γU_1(*warm*)] + T(cool, slow, cool) [R(cool, slow, cool) + γU_1(cool)] or T(cool, fast, cool) [R(cool, fast, cool) + γU_1(cool)]

0.5 * (2+ 1**1*) = 1.5

1 * (1 + 1**2*) = 3 or
0.5* (2 + 1**2*) = 2

FROM WARM:
T(warm, fast, overheated) [R(warm, fast, overheated) + γU_1(*overheated*)] ot
T(warm, slow, cool) [R(warm, slow, cool) + γU_1(cool)]
+
T(warm, slow, warm) [R(warm, slow, warm) + γU_1(warm)]

1* (-10 + 1**0*) = -10 or
0.5* (1 + 1**2*) = 1.5
+
0.5* (1 + 1**1*) = 1

	cool	**warm**	**overheated**
U0	0	0	0
U1	2	1	0
U2	3.5	2.5	0

33. What is the policy based on U2? This step is also called **Policy Extraction.**

    if cool, go fast

    if warm, go slow

34. Suppose there are S states and A actions, what is the time complexity of value iteration per iteration?

    $O(S^2 . A)$

    There are S^2 number of rewards per action

35. Is the value iteration algorithm optimal?

    This is theoretically optimal.

36. So far, we completed the design of value iteration to approximate an optimal policy. Please list out 1-2 disadvantages with Value Iteration?

    - It can be expensive to run.

D. Policy Iteration	start time:

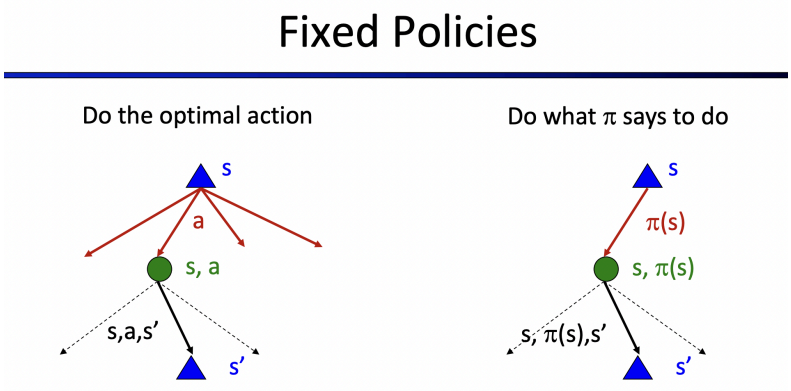
Policy Iteration is an algorithm used to find the optimal policy π* by iteratively evaluating and improving a policy. It consists of two main steps: **policy evaluation** and **policy improvement**:

- **Initialize a policy** π(s) arbitrarily for all states.
- **Policy Evaluation**: Compute the value function U^π(s) for the current policy π.
- **Policy Improvement**: Update the policy π by selecting the action that maximizes the expected value.
- **Repeat Steps 2 & 3** until the policy converges (i.e., no further changes occur).

Put it together in pseudocode as below:

- Initialize $\pi_0(s) = some\ default\ action$ for all s
- for i of policy iteration:
  - *Policy evaluation:*
    - Initialize $V_0^{\pi_i}(s) = 0$ for all s
    - for k of policy evaluation:
      - $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma\, V_k^{\pi_i}(s')\right]$
  - *Policy improvement:*
    - $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$

37. In the picture below, pi is a policy. What is the difference in the choice of action between left and right?



- on left side, we need to make a choice on what path to take. However, on the right, we don't need to choose path, we will just follow whatever is returned by pi.

38. In the step of policy evaluation, we define the utility of a state s under a policy pi as below. What is the recursive relation for the utility in this step? In other words, write the Bellman equation or one-step look-ahead recursive equation for policy evaluation.

- Define the utility of a state s, under a fixed policy π:

  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

- What is the recursive relation (one-step look-ahead / Bellman equation)?
  - Hint: recall Bellman equation for optimal policy:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

input: policy & environment

output: utility

$V^{\pi}(s) = \Sigma_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

39. Based on your equation in Q38, how do we calculate/update the V's for a fixed policy pi? (Hint: look at the pseudocode at the beginning of this section.)

- looping over all the states.
- in value iteration we loop over all the action on each of the state vs on policy iteration we just loop over all the states and they have the policy associated with them

for k of policy evaluation:

- $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$

40. After policy evaluation in Q39, we move to the next step of policy improvement. How do we improve current policy pi after calculating its utilities? (Hint: look at the pseudocode at the beginning of this section.)

Policy improvement:

- $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$

$\pi_{i+1}(s) = \max_a \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$

41. When should we stop improving the policy?

    Compare the actions in the new policy and the current policy. If the actions are the same, we can stop improvement.

    Key value pair in the dictionary should be exactly the same in order to stop the iteration. Dictionary has states with actions as the values.

42. Suppose there are S states and A actions, what is the time complexity of policy iteration per iteration in the policy evaluation step?

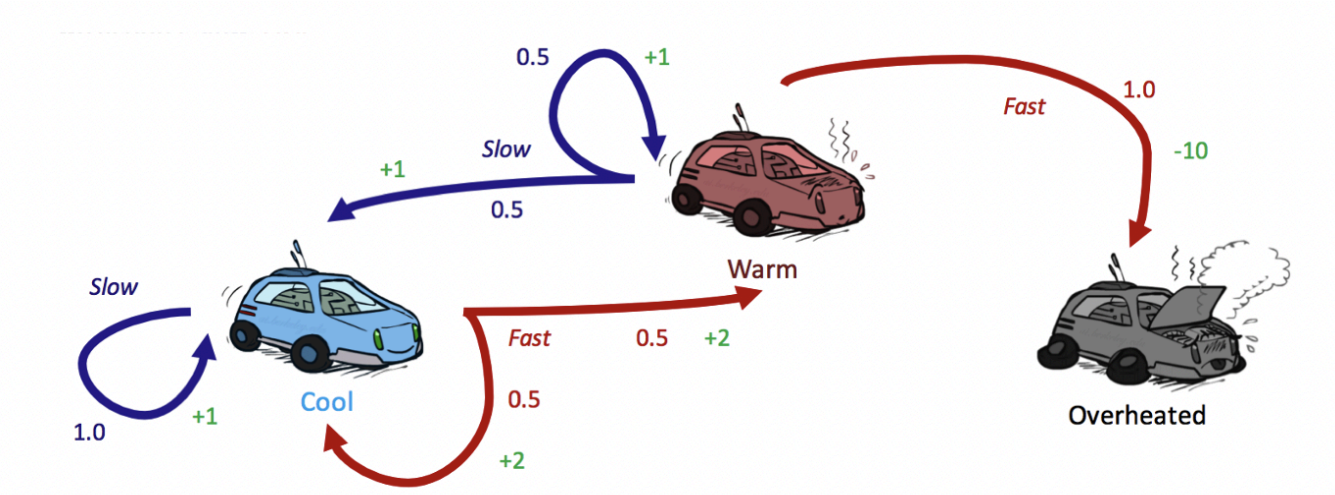
43. Is the policy iteration algorithm optimal?

    Yes.

44. What is the difference between value iteration and policy iteration? Which one works better in which kinds of applications?
    - Policy iteration is faster than value iteration in many cases.
    - Policy iteration can be used when each states have many number of possible actions. Value iteration can be used when there is only a limited number of actions on each state. Because value iteration has to go through all the actions on each state. But policy iteration only has to go through actions for a given state and policy.

Exercise

Consider the motivating example of a racecar:



There are three possible states, S = {cool,warm,overheated}, and two possible actions A = {slow, fast}. Just like in a state-space graph, each of the three states is represented by a node, with edges representing actions. Overheated is a terminal state, since once a racecar agent arrives at this state, it can no longer perform any actions for further rewards (it's a sink state in the MDP and has no outgoing edges). Notably, for nondeterministic actions, there are multiple edges representing the same action from the same state with differing successor states. Each edge is annotated not only with the action it represents, but also a transition probability and corresponding reward.

T(cool, fast, warm) = 0.5
T(cool, slow, cool) = 1.0
T(cool, fast, cool) = 0.5
T(warm, fast, overheated) = 1.0
T(warm, slow, cool) = 0.5
T(warm, slow, warm) = 0.5

R(cool, fast, warm) = +2
R(cool, slow, cool) = +1
R(cool, fast, cool) = +2
R(warm, fast, overheated) = -10
R(warm, slow, cool) = +1
R(warm, slow, warm) = +1

45. We initialize all utilities as 0 and the initial dummy policy is slow for every state. Fill the table below for policy evaluation.

policy =

cool (π(cool)): slow

warm ((π(warm))): slow

overheated (π(overheated)): slow

$V^{\pi}(s) = \Sigma\ T(s, \pi(s), s')\ [R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

from cool:

$V^{\pi}(s) = \Sigma\ T(s, \pi(s), s')\ [R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

$= T(cool, \pi(cool), warm)\ [R(cool, \pi(cool), warm) + 0 * V^{\pi}(warm)] +$
$T(cool, \pi(cool), cool)\ [R(cool, \pi(cool), cool) + 0 * V^{\pi}(cool)]$

$= T(cool, slow, warm)\ [R(cool, slow, warm) + 0 * V^{\pi}(warm)] +$
$T(cool, slow, cool)\ [R(cool, slow, cool) + 0 * V^{\pi}(cool)]$

$= 0*(0+0) + 1.0*(1+0) = 1$

form warm:

$V^{\pi}(s) = \Sigma\ T(s, \pi(s), s')\ [R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

$= T(warm, \pi(warm), warm)\ [R(warm, \pi(warm), warm) + \gamma V^{\pi}(warm)] +$
$T(warm, \pi(warm), cool)\ [R(warm, \pi(warm), cool) + \gamma V^{\pi}(cool)] +$
$T(warm, \pi(warm), oh)\ [R(warm, \pi(warm), oh) + \gamma V^{\pi}(oh)]$

$= T(warm, slow, warm)\ [R(warm, slow, warm) + \gamma V^{\pi}(warm)]$ or
$T(warm, slow, cool)\ [R(warm, slow, cool) + \gamma V^{\pi}(cool)] +$
$T(warm, slow, oh)\ [R(warm, slow, oh) + \gamma V^{\pi}(oh)]$

$= 1 * (1 + 0) + 0\ (0 + 0) = 1$

	cool	**warm**	**overheated**
U0	0	0	0
U1	1	1	0
U2	1*(1+(1*1)) = 2	0.5*(1+(1*1)) + 0.5*(1+(1*1)) = 2	0

46. Now run the step of policy improvement to give a better policy.

Policy improvement:

- $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$

	cool	warm
slow	1(1+2)	.5(1+2)+.5(1+2)
fast	0.5(2+2) + 0.5(2+2)=4	-10