Lab 14
Reinforcement Learning

Datalab

Department of Computer Science,

National Tsing Hua University, Taiwan

Outline

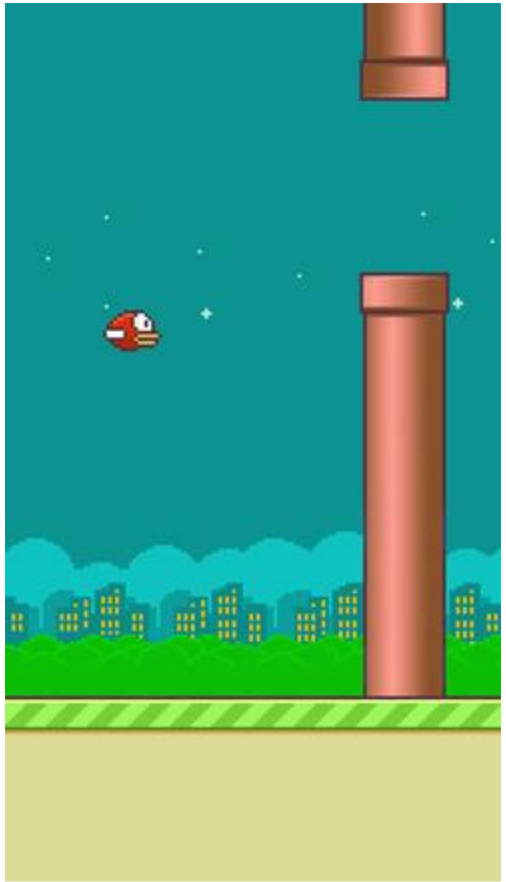
- Homework
- Markov Decision Process (MDP)
  - Value Iteration
  - Policy Iteration
- Q-Learning & SARSA

Outline

- Homework
- MDP(value iteration & policy iteration)
- Q-Learning & SARSA

Homework

- Train an agent to play Flappy Bird game(SARSA)



Install PLE and Pygame

- Clone the repo

```
$ git clone https://github.com/ntasfi/PyGame-Learning-Environment
Cloning into 'PyGame-Learning-Environment'...
remote: Enumerating objects: 1118, done.
remote: Total 1118 (delta 0), reused 0 (delta 0), pack-reused 1118
Receiving objects: 100% (1118/1118), 8.06 MiB | 800.00 KiB/s, done.
Resolving deltas: 100% (592/592), done.
```

- Install PLE(in the PyGame-Learning-Environment folder)
  - cd PyGame-Learning-Environment
  - pip install –e .

```
$ pip install -e .
Obtaining file:///E:/DL/Lab/RL/PyGame-Learning-Environment
Requirement already satisfied: numpy in c:\users\vincent\anaconda3\lib\site-pack
ages (from ple==0.0.1) (1.16.4)
Requirement already satisfied: Pillow in c:\users\vincent\anaconda3\lib\site-pac
kages (from ple==0.0.1) (6.1.0)
Installing collected packages: ple
  Found existing installation: ple 0.0.1
    Uninstalling ple-0.0.1:
      Successfully uninstalled ple-0.0.1
  Running setup.py develop for ple
Successfully installed ple
```

- pip install pygame

Homework

- What you should do:
  - Change the update rule from Q-learning to SARSA (**with the same episodes**).
  - Give a brief report to discuss the result (compare Q-learning with SARSA based on the game result).
- Remind
  - Only need CPU resources.
  - It will take you more than 13 hours to train, please reserve enough time.

Homework

- Precautions:
  - If you encounter this problem, just stop.
    - It means your bird plays well and the recorded frames is too long to save.

```
~\Anaconda3\lib\site-packages\moviepy\video\io\html_tools.py in html_embed(clip, filetype, maxduration, rd_kwargs, center, **html_k
wargs)
    105
    106         return html_embed(filename, maxduration=maxduration, rd_kwargs=rd_kwargs,
--> 107                           center=center, **html_kwargs)
    108
    109     filename = clip

~\Anaconda3\lib\site-packages\moviepy\video\io\html_tools.py in html_embed(clip, filetype, maxduration, rd_kwargs, center, **html_k
wargs)
    140         if duration > maxduration:
    141             raise ValueError("The duration of video %s (%.1f) exceeds the 'maxduration' "%(filename, duration)+
--> 142                              "attribute. You can increase 'maxduration', by passing 'maxduration' parameter"
    143                              "to ipython_display function."
    144                              "But note that embedding large videos may take all the memory away !")

ValueError: The duration of video __temp__.mp4 (129.8) exceeds the 'maxduration' attribute. You can increase 'maxduration', by passing 'max
duration' parameterto ipython_display function.But note that embedding large videos may take all the memory away !
```

Homework

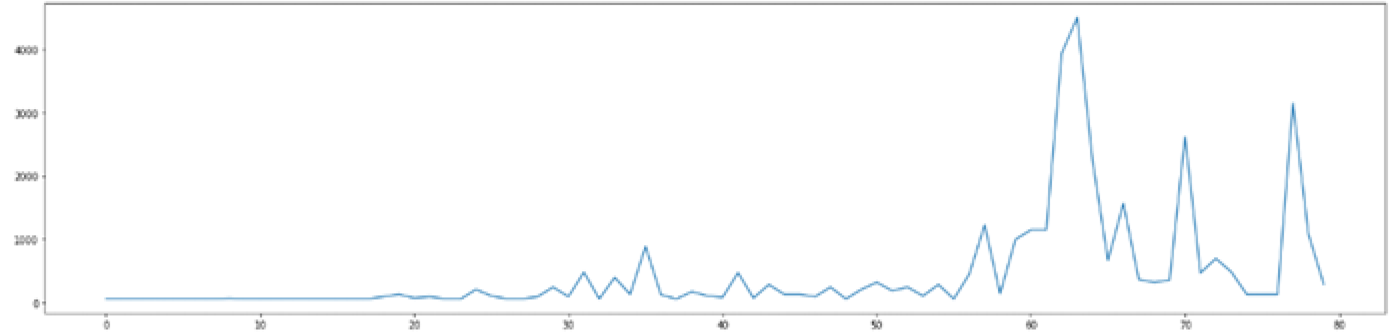
Requirements

- Write a brief report in the notebook
- Upload both ipynb and mp4 to google drive
  - Lab14_{student_id}.ipynb (90%)
  - Lab14_{student_id}.mp4 (10%)
- Share your drive's link via eeclass
  - Please make sure that TA can access your google drive!!!
    (Or you will get 0 on this lab!)

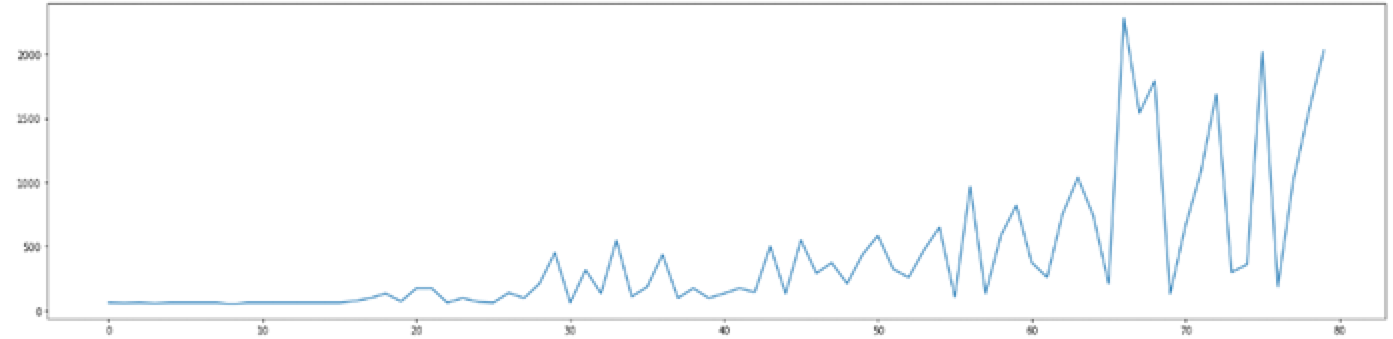
- Deadline: 2023-12-11(Wed) 23:59

Homework

- Requirement (report):
  - You can compare life time or reward against training episodes.



Q-learning



SARSA

Outline

- Homework
- Markov Decision Process (MDP)
  - Value Iteration
  - Policy Iteration
- Q-Learning & SARSA

Markov Decision Process (MDP)

- A MDP is defined by

S	A	P	R	γ	H
State space	Action space	Transition Probability	Reward	Discount Factor	Horizon

小吃	資電	排球
綜二	台達	籃球
總圖	工三 	西門

S = { 小吃, 資電, 排球, 綜二, 台達, 籃球, 總圖, 工三, 西門 }
A = { 上, 下, 左, 右 }
P = no noise
R(工三) = 0, R(others) = -1
r = 1
H = inf

We have a MDP model, then?

Goal - Find the Optimal Policy

- If the agent follow the optimal policy, it will get maximal total reward
- We can solve it via these two algorithms
  - Value Iteration
  - Policy Iteration

Outline

- Homework
- Markov Decision Process (MDP)
  - Value Iteration
  - Policy Iteration
- Q-Learning & SARSA

Value Iteration

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi^*(\boldsymbol{s})$'s for all $\boldsymbol{s}$'s

For each state $\boldsymbol{s}$, initialize $V^*(\boldsymbol{s}) \leftarrow 0$;
repeat
  foreach $\boldsymbol{s}$ **do**
    $V^*(\boldsymbol{s}) \leftarrow \max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V^*(\boldsymbol{s}')]$;
  end
until $V^*(\boldsymbol{s})$*'s converge*;
foreach $\boldsymbol{s}$ **do**
  $\pi^*(\boldsymbol{s}) \leftarrow \arg\max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V^*(\boldsymbol{s}')]$;
end

 小吃	資電	排球
綜二	台達	籃球
總圖	工三 	西門

Input: MDP $(\mathbb{S}, \mathbb{A}, P, R, \gamma, H \to \infty)$

Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s) \leftarrow 0$;

repeat

  foreach s **do**

    $V^*(s) \leftarrow \max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;

  end

until $V^*(s)$*'s converge*;

foreach s **do**

  $\pi^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;

end

S = { 小吃, 資電, 排球, 綜二, 台達, 籃球, 總圖, 工三, 西門 }
A = { 上, 下, 左, 右 }
P = no noise
R(工三) = 0, R(others) = -1
r = 1
H = inf

After Initialization

小吃 0	資電 0	排球 0
綜二 0	台達 0	籃球 0
總圖 0	工三 0 	西門 0

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s) \leftarrow 0$;
repeat
  foreach s **do**
    $V^*(s) \leftarrow \max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
  end
until $V^*(s)$*'s converge*;
foreach s **do**
  $\pi^*(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
end

After Iteration 1

小吃 -1	資電 -1	排球 -1
綜二 -1	台達 -1	籃球 -1
總圖 -1	工三 0	西門 -1

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s) \leftarrow 0$;
repeat
  foreach s **do**
    $V^*(s) \leftarrow \max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
  end
until $V^*(s)$*'s converge*;
foreach s **do**
  $\pi^*(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
end

V(台達) = Reward + V(資電) = -1 + 0

V(台達) = Reward + V(綜二) = -1 + 0

V(台達) = Reward + V(工三) = -1 + 0

V(台達) = Reward + V(籃球) = -1 + 0

After Iteration 2

小吃 -2	資電 -2	排球 -2
綜二 -2	台達 -1	籃球 -2
總圖 -1	工三 0	西門 -1

Input: MDP $(\mathbb{S}, \mathbb{A}, P, R, \gamma, H \to \infty)$
Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s) \leftarrow 0$;
repeat
  foreach s **do**
    $V^*(s) \leftarrow \max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
  end
until $V^*(s)$*'s converge*;
foreach s **do**
  $\pi^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
end

$\max_a$
V(台達) = Reward + V(資電) = -1 + -1
V(台達) = Reward + V(綜二) = -1 + -1
V(台達) = Reward + V(工三) = -1 + 0
V(台達) = Reward + V(籃球) = -1 + -1

After Iteration 3

小吃 -3	資電 -2	排球 -3
綜二 -2	台達 -1	籃球 -2
總圖 -1	工三 0	西門 -1

Input: MDP $(\mathbb{S}, \mathbb{A}, P, R, \gamma, H \rightarrow \infty)$
Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s) \leftarrow 0$;
repeat
  foreach s **do**
    $V^*(s) \leftarrow \max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
  end
until $V^*(s)$'s *converge*;
foreach s **do**
  $\pi^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
end

V(小吃) = V(排球) = -1 + -2 = -3

After Iteration 4

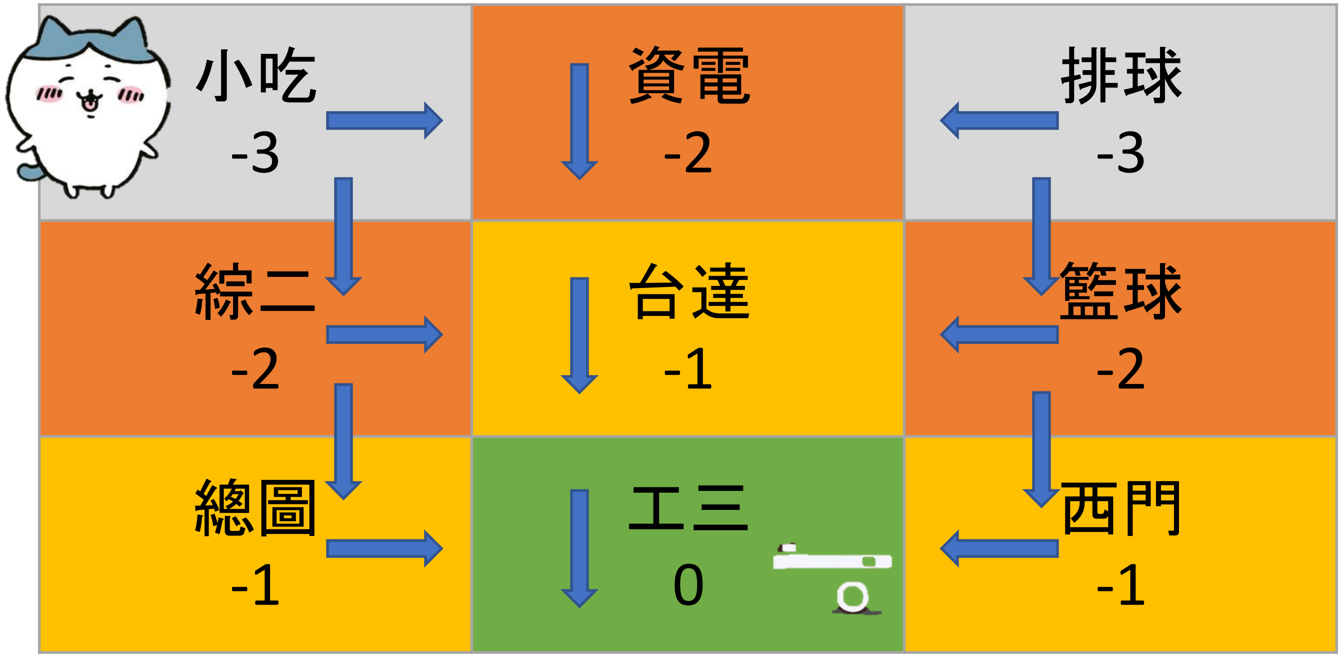
小吃 -3	資電 -2	排球 -3
綜二 -2	台達 -1	籃球 -2
總圖 -1	工三 0 	西門 -1

Input: MDP $(\mathbb{S}, \mathbb{A}, P, R, \gamma, H \rightarrow \infty)$
Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s) \leftarrow 0$;
repeat
  foreach s **do**
    $V^*(s) \leftarrow \max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
  end
until $V^*(s)$'s *converge*;
foreach s **do**
  $\pi^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s;a)[R(s,a,s') + \gamma V^*(s')]$;
end

Iteration 4 = Iteration 3

Converge !



Input: MDP $(\mathbb{S},\mathbb{A},\mathrm{P},R,\gamma,H\to\infty)$
Output: $\pi^*(s)$'s for all s's

For each state s, initialize $V^*(s)\leftarrow 0$;
repeat
  foreach s **do**
    $V^*(s)\leftarrow \max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s')+\gamma V^*(s')]$;
  end
until $V^*(s)$*'s converge*;
foreach s **do**
  $\pi^*(s)\leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s')+\gamma V^*(s')]$;
end

Now we have the optimal policy!

Outline

- Homework
- Markov Decision Process (MDP)
  - Value Iteration
  - Policy Iteration
- Q-Learning & SARSA

Policy Iteration

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

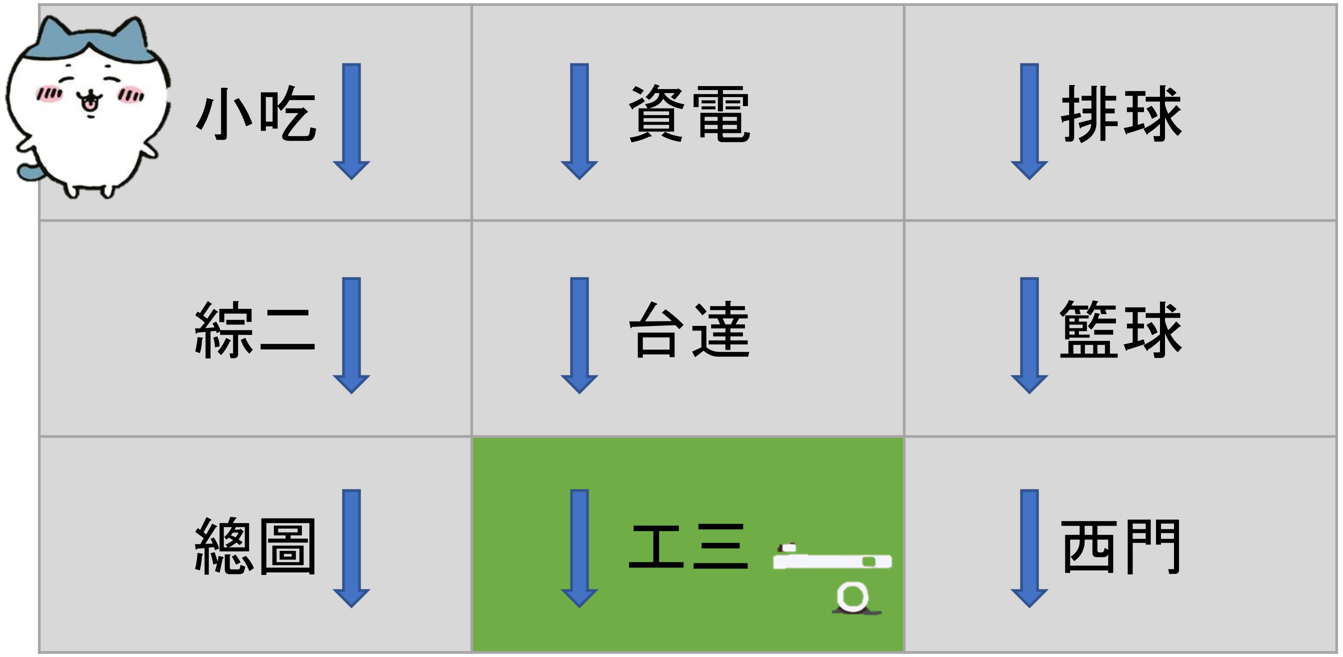
For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat Policy evaluation
    foreach s **do**
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$ *'s converge*;
  foreach s **do** Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$ *'s converge*;

小吃	資電	排球
綜二	台達	籃球
總圖	工三	西門

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do**   Policy evaluation
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do**   Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

S = { 小吃, 資電, 排球, 綜二, 台達, 籃球, 總圖, 工三, 西門 }
A = { 上, 下, 左, 右 }
P = no noise
R(工三) = 0, R(others) = -1
r = 1
H = inf



Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$

Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;

repeat

  For each state s, initialize $V_\pi(s) \leftarrow 0$;

  repeat

    foreach s **do**

Policy evaluation

      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;

    end

  until $V_\pi(s)$*'s converge*;

  foreach s **do**

Policy improvement

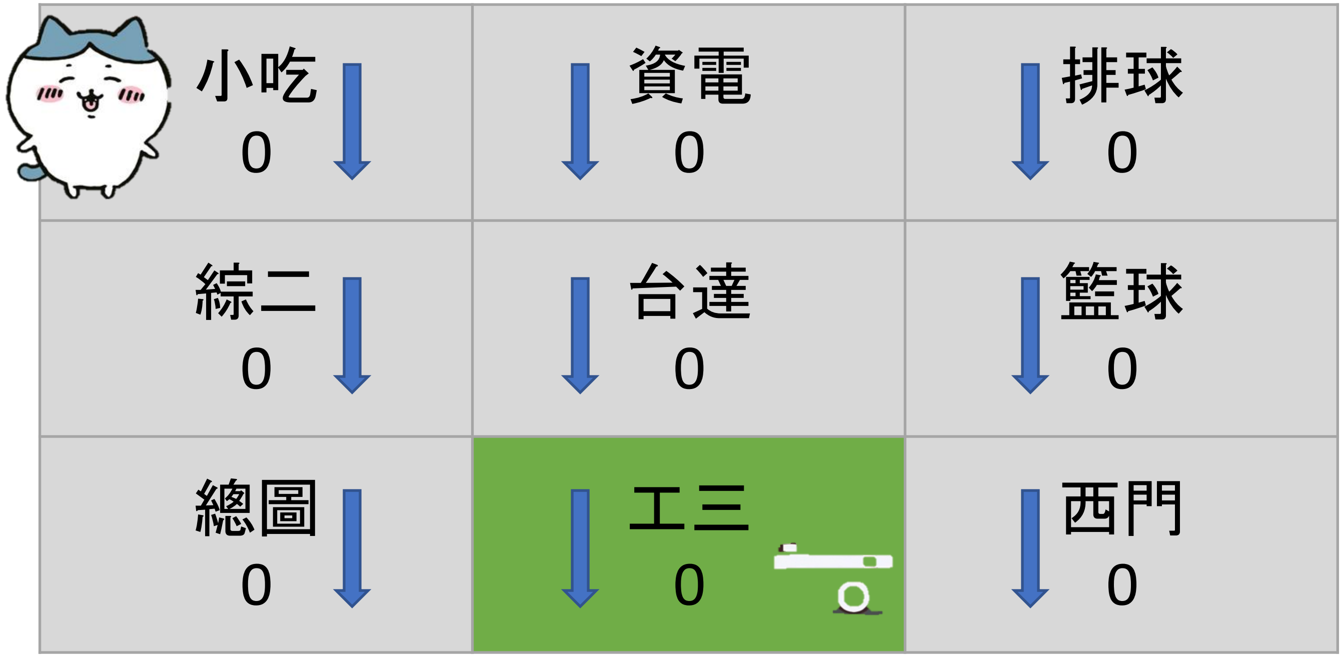
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;

  end

until $\pi(s)$*'s converge*;

Random initialize a policy

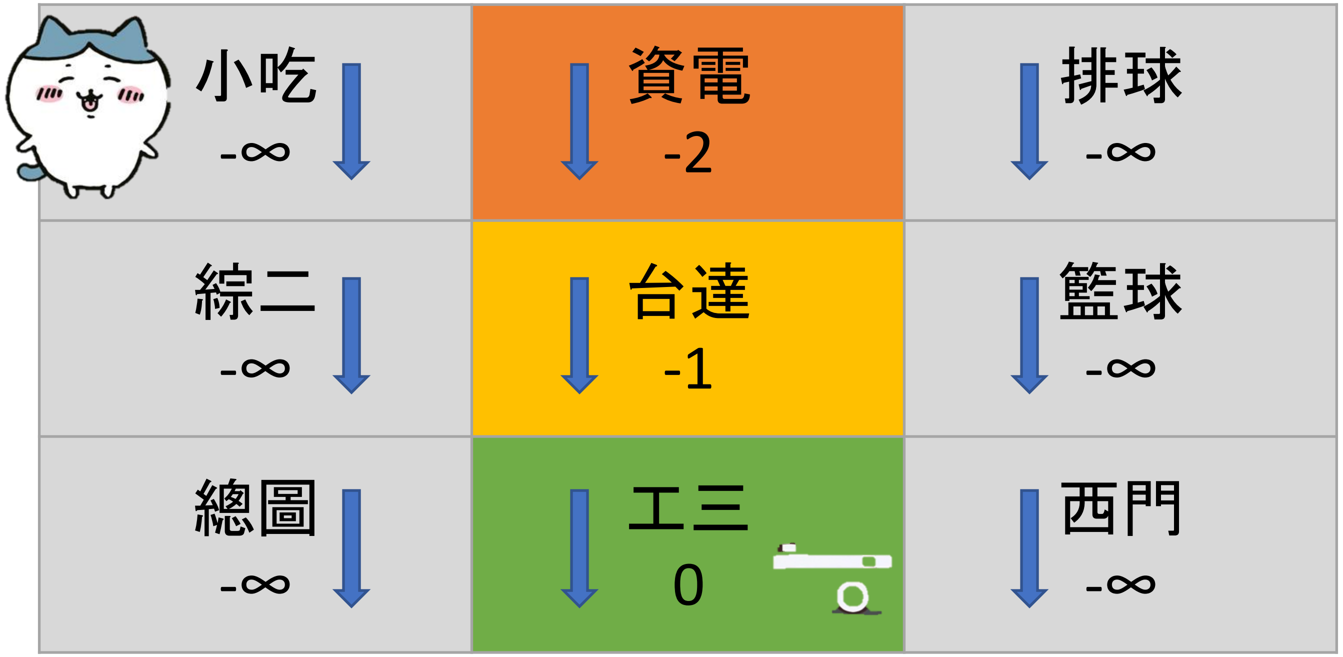
Let’s say all goes down!



Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat Policy evaluation
    foreach s **do**
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do** Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

After initialization of V_π

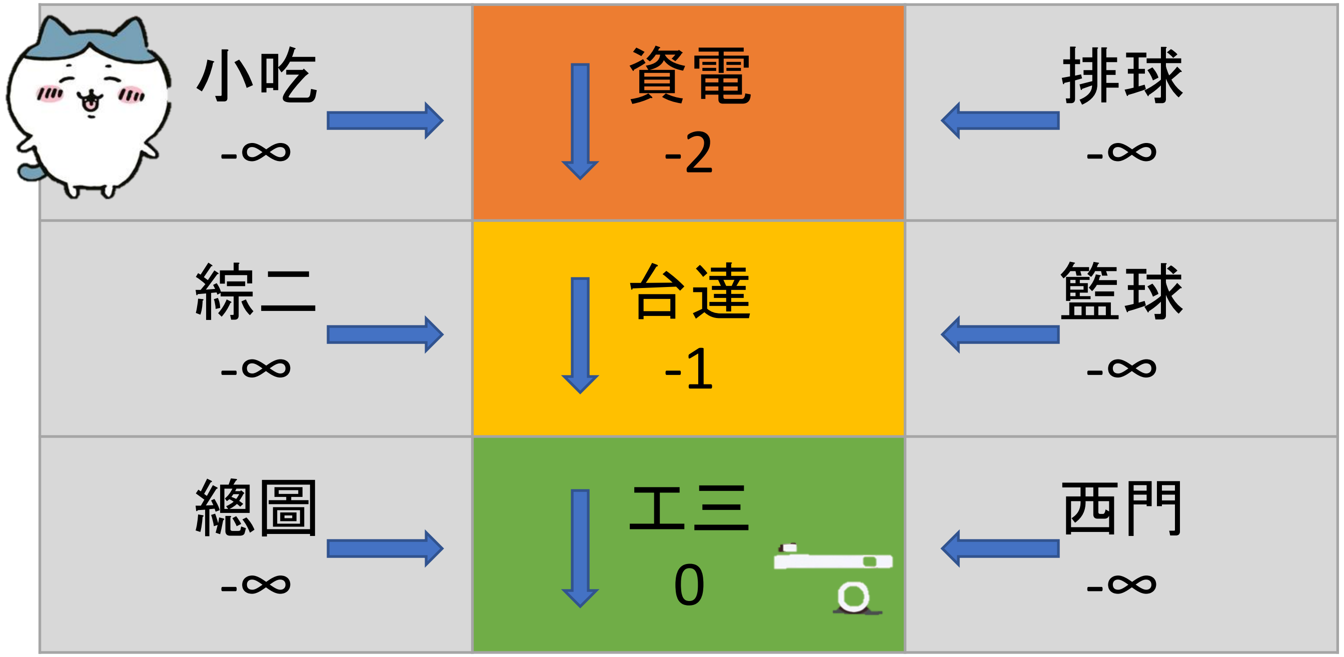


Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \rightarrow \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do** ← Policy evaluation
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do** ← Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

After Policy Evaluation

After Policy Improvement



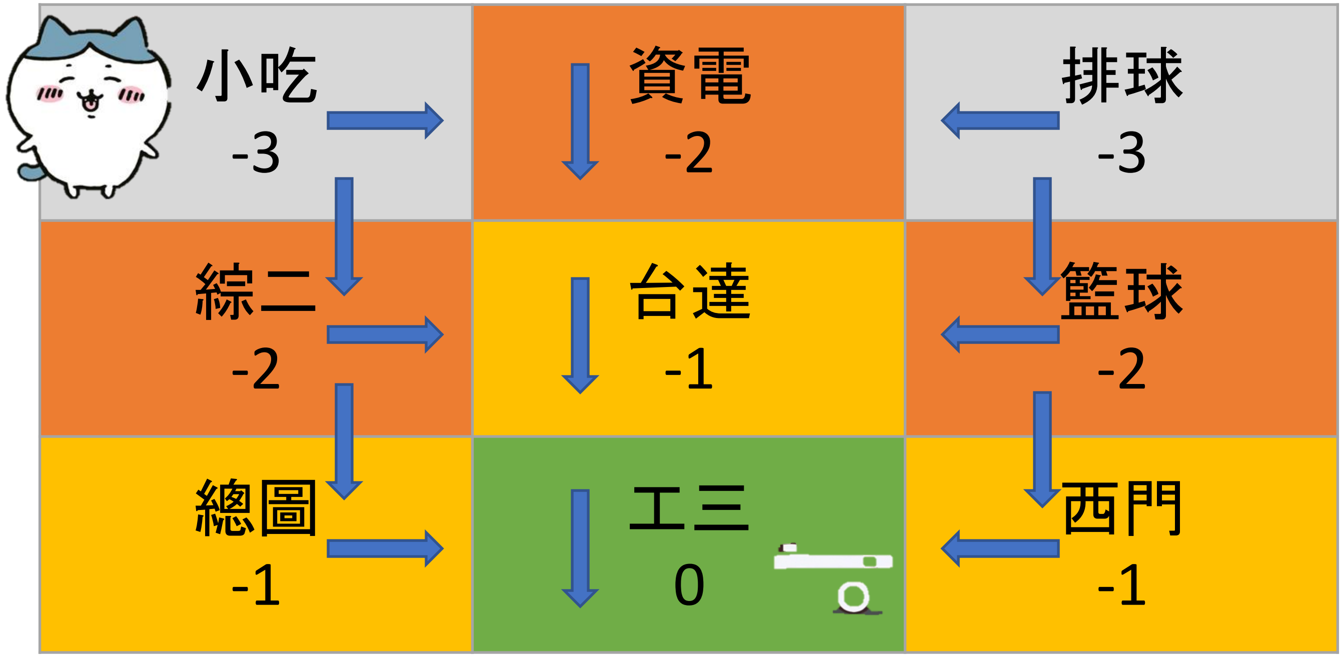
Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do**   Policy evaluation
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do**   Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

V(籃球) = reward + V(西門) = -1 + -∞

$\arg\max_a$ V(籃球) = reward + V(台達) = -1 + -1

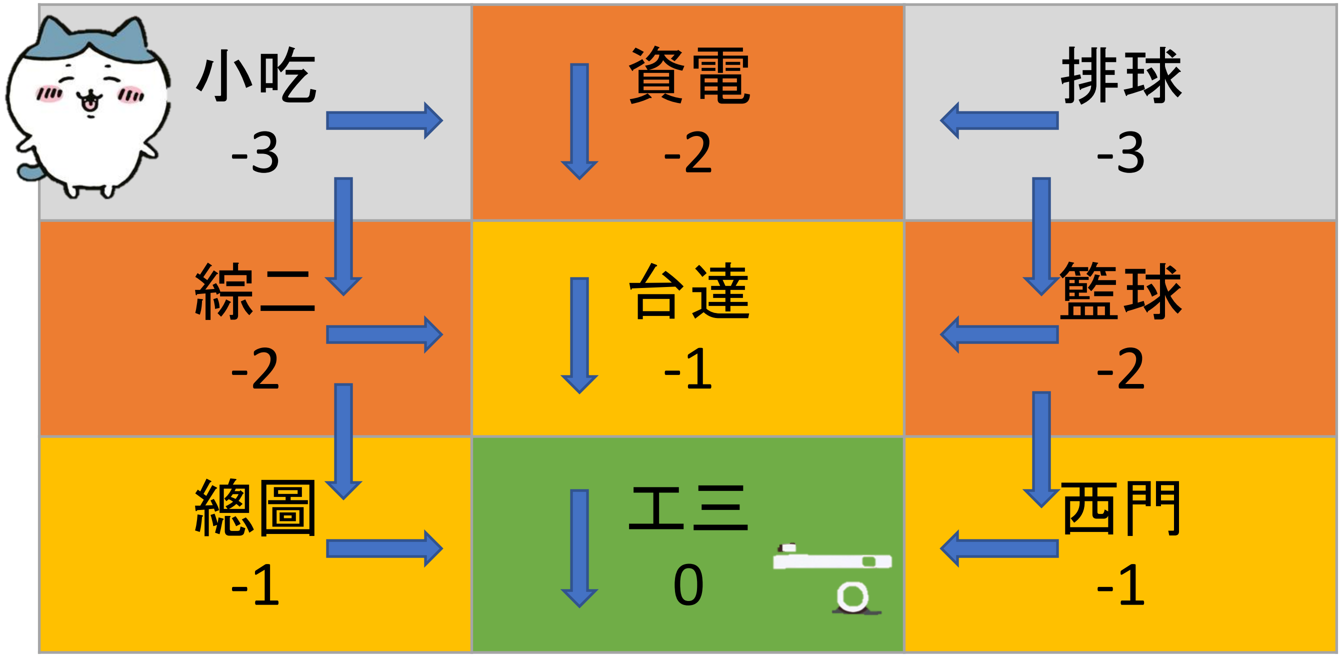
V(籃球) = reward + V(排球) = -1 + -∞



Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do** ← Policy evaluation
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do** ← Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

Policy Evaluation Again!



Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do** — Policy evaluation
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do** — Policy improvement
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

Policy Improvement.
Nothing Changed!
Converge!!

Did agent Interact with the Environment?

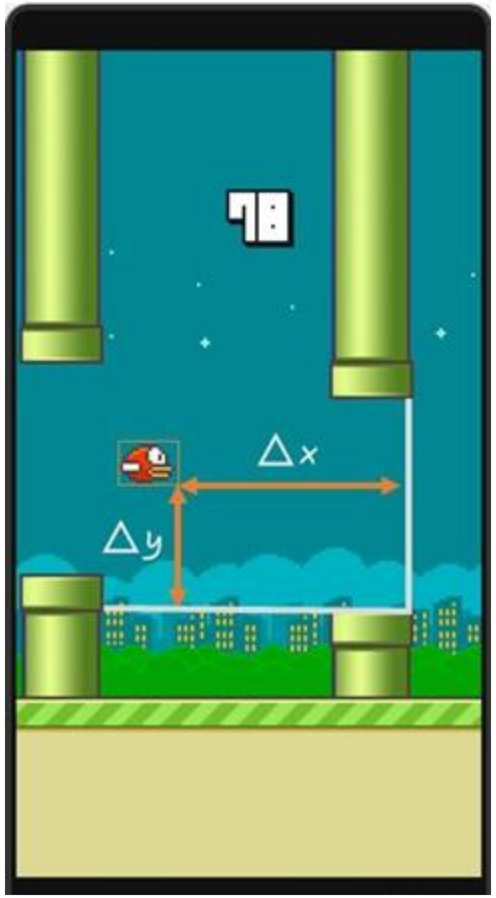
- No ! We model every transition and every reward
- But it is impossible to solve more complex problems like Flappy Bird
- We need model-free algorithms
  - Q-Learning
  - SARSA

Outline

- Homework
- Markov Decision Process (MDP)
  - Value Iteration
  - Policy Iteration
- Q-Learning & SARSA

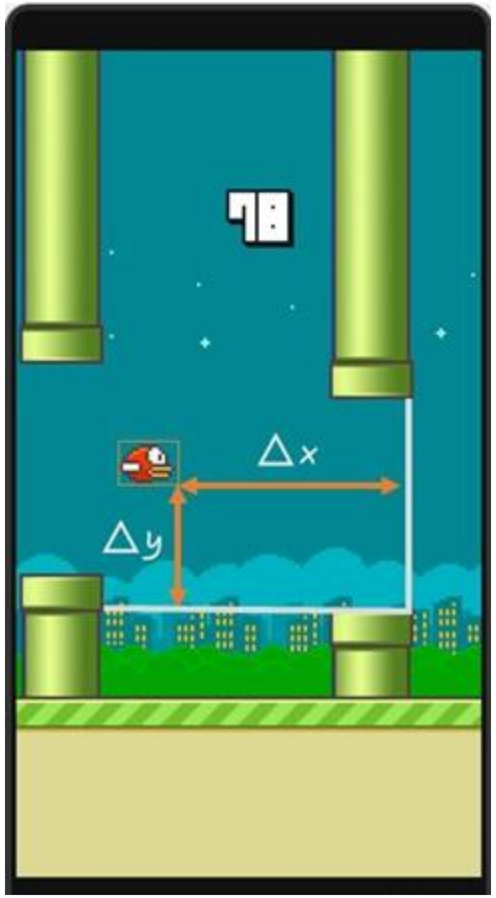
Q-Learning

- Flappy bird



Q-Learning

- Flappy Bird
- States: $(\Delta x, \Delta y)$
- Actions: { fly, none }
- Reward:
  - +1: pass through a pipe
  - -5: die



ref: https://www.zhihu.com/question/26408259

Q-Learning

- Q-table(finite):

狀態	飛	不飛
$(\Delta x_1, \Delta y_1)$	1	20
$(\Delta x_1, \Delta y_2)$	20	-100
$\dots$	$\dots$	$\dots$
$(\Delta x_m, \Delta y_{n-1})$	-100	2
$(\Delta x_m, \Delta y_n)$	50	-200

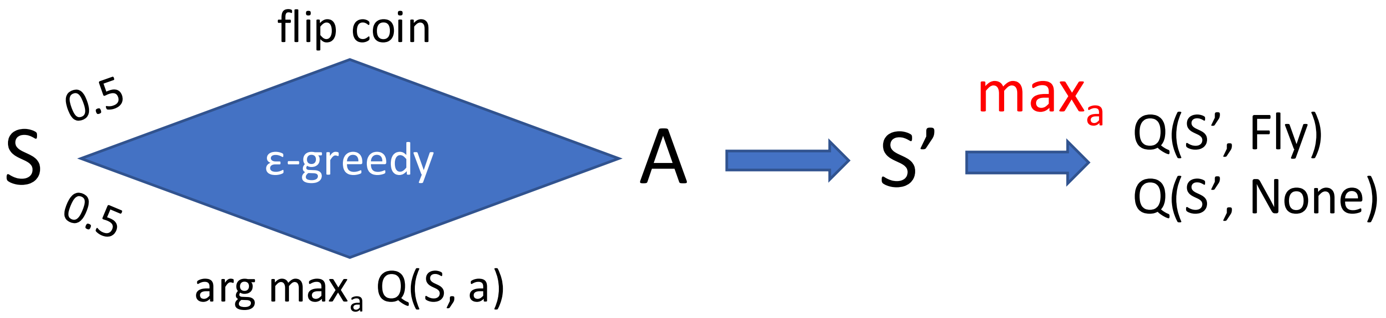
- Update rule: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

Q-Learning

- Algorithm

Q-learning: An off-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal



SARSA

- Algorithm

Sarsa: An on-policy TD control algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

SARSA

- Q-table(finite):

狀態	飛	不飛
$(\Delta x_1, \Delta y_1)$	1	20
$(\Delta x_1, \Delta y_2)$	20	-100
$\cdots$	$\cdots$	$\cdots$
$(\Delta x_m, \Delta y_{n-1})$	-100	2
$(\Delta x_m, \Delta y_n)$	50	-200

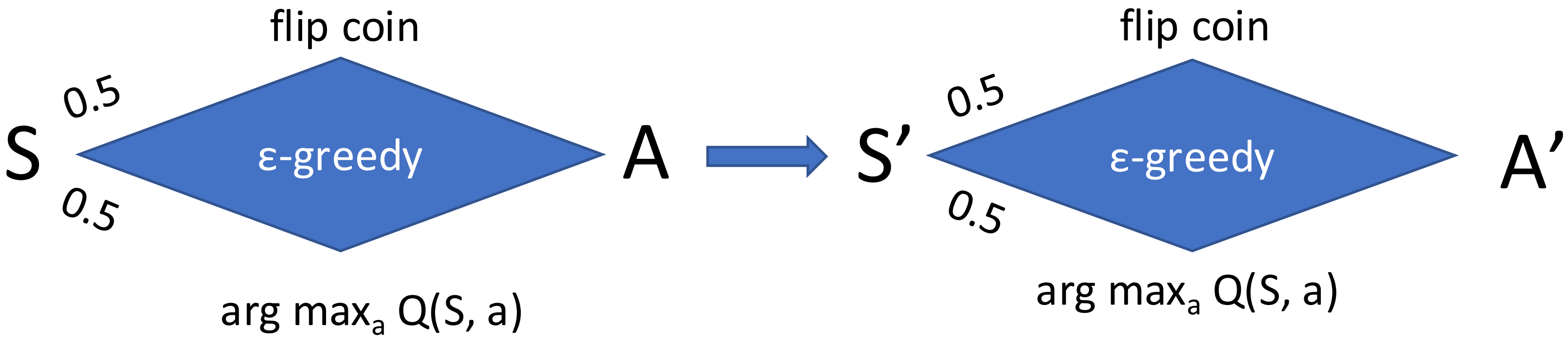
- Update rule: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

SARSA

- Algorithm

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal



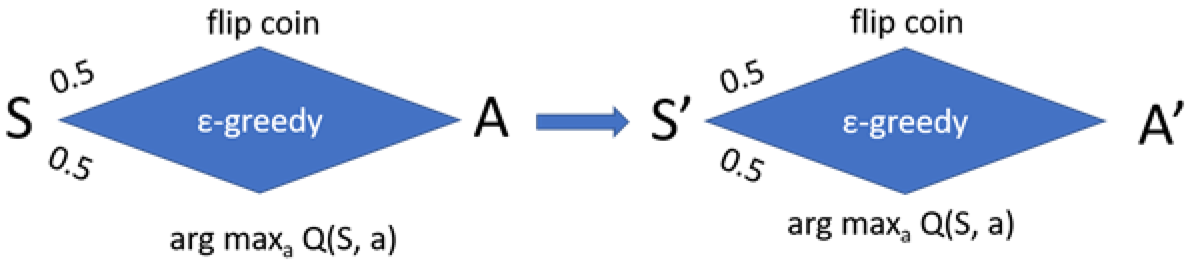
Q-Learning VS. SARSA

- Difference

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S ②
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

③ ①

Figure 6.9: Sarsa: An on-policy TD control algorithm.

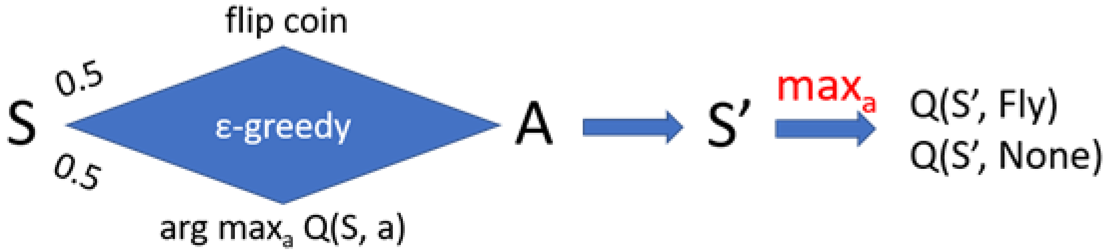


Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S ②
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$; **what about A'?**
  until S is terminal

① **this is like following a greedy policy (e.g. ε=0, NO exploration)**

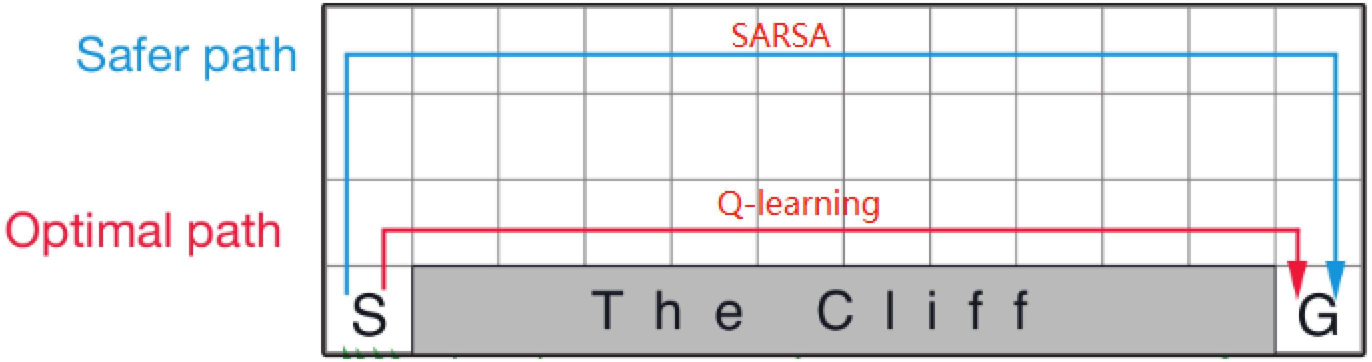
③

Figure 6.12: Q-learning: An off-policy TD control algorithm.



Q-Learning VS. SARSA

- Cliff Walking



Thanks! Be a Happy Bird