Basic Knowledge:

Core Components:

State. Action. Award

Unchanged once set

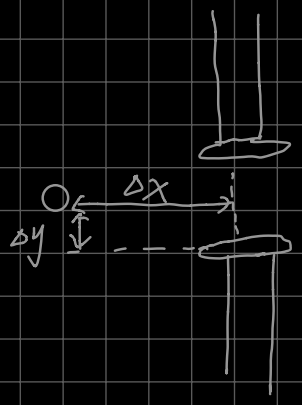
Q-learning:

Q: action-utility function, 用于评价在状态 s 下采取动作 a 的优劣。 ⇒ 相当于 agent 的记忆，随着训练过程而改变。(即 Q-table)

f(x) = y

Entity: Flappy Bird

State ⇒ $(\Delta x, \Delta y)$



Action ⇒ fly. fall

Award ⇒ alive through 1 tube: +100, dead: -1000

Q/action-utility function ⇒ m×n table, each grid in the table represents utility value of an action fly/fall upon a state $(\Delta x_i, \Delta y_j)$.

State	fly	fall
$(\Delta x_1, \Delta y_1)$	+1	+20
$(\Delta x_1, \Delta y_2)$	-100	2
⋮		
$(\Delta x_m, \Delta y_n)$	50	-200

⇒ Q

Train goal: To get a perfect table Q, which makes the bird get maximum utility on current state and will be alive forever. How to Train ⇒

```
def train():
    Init Q = {};
    while Q is still Diverged:
        Init state S ← S0
        while S != Dead:
            Get action a through strategy π(S0)
            Get award R(s, a) and new state S'
            #update Q[S,A]
            Q[S,A] ← (1-α)*Q[S,A] + α*(R(s,a) + γ*max_a' Q[S',a'])
            # update State
            S = S'
```

Get core components — strategy → according to Q. choose max utility.

Update — past training: $(1-\alpha)*Q[S,A]$; learning rate: α; current utility: $R(s,a)$; discount factor: γ; memorial utility: $\max_{a'} Q[S',a']$

△ α越大，过去训练保留的效果越少

γ越大，agent越重视过去的经验，而忽视眼前的利益。

△ 缺点：① 适用的 State space、action space 非常小

② 没有预测能力/泛化能力 ⇒ 当一个 states 未出现过，Q learning 无法处理

How to solve ⇒

Solve ①:

Policy Based method.

Solve ②:

DQN (Deep Q-learning Network)