Brief Introduction to Reinforcement Learning

Examples in Q-learning

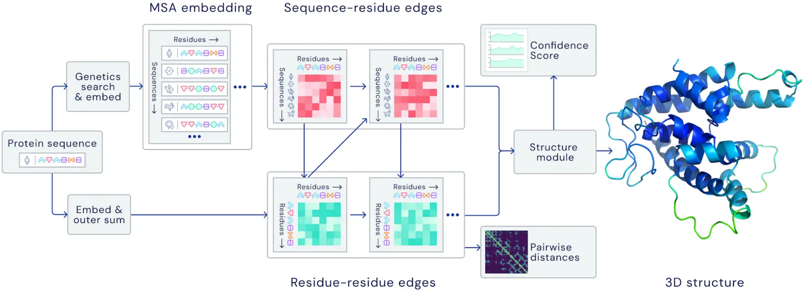
Eric Qu

Duke Kunshan University

December 15, 2020

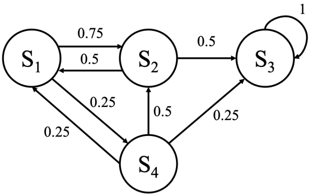
AlphaFold

- Recently, Google has made huge progress in protein folding.
- The program AlphaFold could shorten the time to work out certain structure of proteins given certain amino acids sequences from a few weeks of experiments to a few hours of computation.
- This is "a solution to a 50-year-old grand challenge in biology".
- The algorithm they use is - **Reinforcement Learning**



Markov Process (Recall)

- Markov Property
  - $$\mathbb{P}(S_{t+1}|S_t) = \mathbb{P}(S_{t+1}|S_1, S_2, \ldots, S_t)$$
- Markov Process (Markov Chain)
  - Memoryless stochastic process defined by $< \mathcal{S}, \mathcal{P} >$
  - $\mathcal{S}$ State Space
  - $\mathcal{P}$ Transition Probability Matrix, $\mathcal{P}_{ss'} = P(S_{t+1} = s'|S_t = s)$
- Example:



$$\mathcal{P} = \begin{bmatrix} 0 & 0.75 & 0 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 \end{bmatrix}$$

Markov Reward Process

Definition

- Defined by $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$
- Reward $\mathcal{R}$
  - R_t: Reward in time t.
  - Reward in state s is: In some time t at state s, the expected reward at $t+1$.
$$R_s = \mathbb{E}[R_{t+1} | S_t = s]$$
  - Why $t+1$? It is defined to get a reward when leaving a state.
- Discount factor $\gamma \in [0, 1]$
- Return
  - Return G_t of a time t is the sum of discounted reward after t.
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}$$
  - γ: immediate return or long-term return

Markov Reward Process

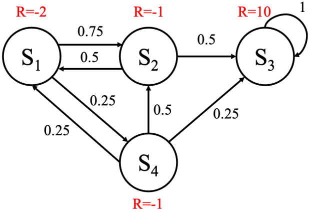
Value Function

- Value Function
  - Expected return starting from state s.

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

- Example

Let $\gamma = 0.5$



$S_1 \rightarrow S_2 \rightarrow S_1 \rightarrow S_4 \rightarrow S_3$:
$G_1 = -1 + 0.5(-2) +$
$(0.5)^2(-1) + (0.5)^3 10$

Markov Reward Process

Ballman Function

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[G_{t+1}|S_{t+1},S_t]|S_t] &= \mathbb{E}[G_{t+1}|S_t] \\
\mathbb{E}[\mathbb{E}[G_{t+1}|S_{t+1},S_t]|S_t] &= \mathbb{E}[\mathbb{E}[G_{t+1}|S_{t+1}]|S_t] \\
&= \mathbb{E}[v(s+1)|S_t] \\
\mathbb{E}[G_{t+1}|S_t] &= \mathbb{E}[v(s+1)|S_t]
\end{aligned}$$

$$\begin{aligned}
v(s) &= \mathbb{E}[G_t|S_t=s] \\
&= \mathbb{E}[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\cdots|S_t=s] \\
&= \mathbb{E}[R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\cdots)|S_t=s] \\
&= \mathbb{E}[R_{t+1}+\gamma G_{t+1}|S_t=s] \\
&= \mathbb{E}[R_{t+1}+\gamma v(S_{t+1})|S_t=s]
\end{aligned}$$

- Ballman function: $v(s)=\mathbb{E}[R_{t+1}+\gamma v(S_{t+1})|S_t=s]$
- Two parts: the immediate reward R_{t+1}, and the discounted value of successor state.
- Also:

$$v(s) = R_s + \gamma \sum_{s'\in\mathcal{S}} P_{ss'} v(s')$$

Markov Decision Process

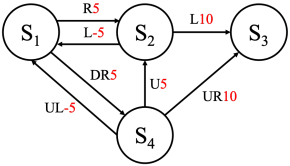
Definition

- A Markov Decision Process is defined by $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$
- $\mathcal{A}$ is the Action Space. Here, each $\mathcal{P}$ and $\mathcal{R}$ is corresponding to an action a:

$$\mathcal{P}^a_{ss'} = P(S_{t+1} = s' | S_t = s, A_t = a)$$

$$\mathcal{R}^a_s = \mathbb{E}[\mathcal{R}_{t+1} | S_t = s, A_t = a]$$

- Example



Action Space:
$\mathcal{A} = \{R, L, U, D, UL, UR, DL, DR\}$
Each action is corresponding to a reward.

Markov Decision Process

Policy

- **Policy**
  - Policy π describes all possible behavior of the agent. $\pi(a|s)$ is the probability of taking action a in state s.

$$\pi(a|s) = P(A_t = a|S_t = s)$$

  - Given a $M =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ and a policy π, then $S_1, S_2, \cdots$ is a Markov Process $< \mathcal{S}, \mathcal{P}^\pi >$, with,

$$\mathcal{P}^\pi_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{P}^a_{ss'}$$

$S_1, R_1, S_2, R_2 \cdots$ is a Markov Decision Process $< \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma >$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{R}^a_s$$

Markov Decision Process

Value Funtion

- **Value Function**
  - **State Value Function**: The expected return staring from state s, following policy π.

    $$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

  - **State-action Value Function**: The expected return staring from state s, take action a, following policy π.

    $$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

  - $$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

    $$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Markov Decision Process

Bellman Expection Function

- We could combine them:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

- **Bellman Expection Function**

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Markov Decision Process

Optimal Value Funtion

- **Optimal State Value Function**: The maximum state value function over all policies.

$$v_*(s) = \max_{\pi} v_\pi(s)$$

- **Optimal State-action Value Function**: The maximum state-action value function over all policies.

$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

- **Optimal Policy**: For every MDP, there exists an optimal policy π_* that is better or equal to all other policies and achieve both optimal value functions: $v_{\pi_*}(s) = v_*(s), q_{\pi_*}(s, a) = q_*(s, a)$ The ultimate goal of Reinforcement Learning is to find the optimal policy.

Markov Decision Process

Bellman Optimality Funtion

- We have

$$v_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- Combine them, then we have the **Bellman Optimality Funtion**

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

Q-learning

Introduction

- Q-learning was a big break through in the early days of Reinforcement Learning.
- For targe policy, Q-learning updates it in a greedy way:

$$\pi(s_{t+1}) = \arg\max_{a'} q(s_{t+1}, a')$$

For every step, choose the action with the max expected return.

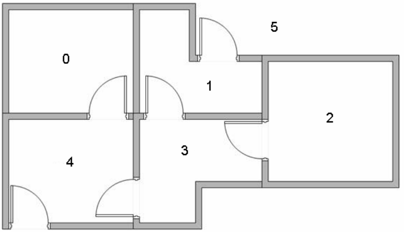
- For $q(s, a)$, it will be updated by Bellman Optimality Function:

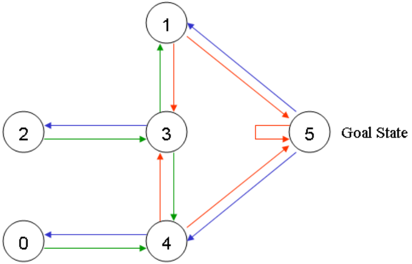
$$q(s, a) \leftarrow (1 - \alpha)q(s, a) + \alpha[\mathcal{R}_s^a + \gamma \max_{a'} q(s', a')]$$

Q-learning

Graph Example

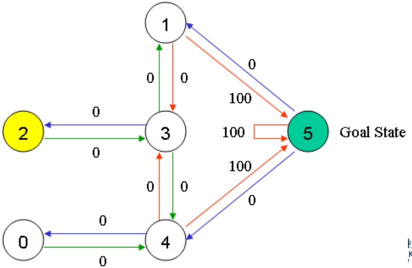
- We could try to train an agent to evacuate from a room.
- The room could be represented by a graph, room as node and door as edge.





- The should first identify the rule of evacuation.
  - We start from a random room and try to get to the outside (5).
  - $\mathcal{S} = \{0,1,2,3,4,5\}, \mathcal{A} = \{0,1,2,3,4,5\}, \mathcal{P}_{ss'}^{a} = \begin{cases} 1 & s' = a \\ 0 & s' \neq a \end{cases}$
- Then, we can construct a reward matrix for the game.
- Note that the construction of reward is of great significance in reinforcement learning.

$$
R = \begin{array}{c c}
 & \textbf{Action} \\
\textbf{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
\end{array}
$$



- Then we set $\gamma = 0.8$ and start Q-learning algorithm.
- During the algorithm, we aim to maintain a Q-matrix that stores all $q(s, a)$ values.
- We initialize the Q-matrix to be all zeros.
- Assume we randomly start at state 1.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{ccc} & & \text{Action} \\ & \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ R = & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

- Assume we randomly start at state 1.
- Then we apply a ϵ-greedy algorithm to choose the next state:
  - From 1, we could reach 3 and 5.
  - ϵ-greedy algorithm is having probability $(1-\epsilon)$ to choose according to $q(s,a)$ and probability ϵ to choose a random action.
  - Purpose of ϵ-greedy: prevent it to stuck in a local maximum.
  - Since the Q table is empty now, we should randomly choose a state from 3 and 5. Assume we choose 5.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

- Assume we randomly start at state 1 and go to state 5.
- After reaching state 5, we should update the Q table:
  - For simplicity concern, we ignore the learning rate α
  - $q(s,a) \leftarrow q(s,a) + \mathcal{R}_s^a + \gamma \max_{a'} q(s',a')$
  - From 5, we could go to state 1, 4 or 5. But the q-value of all these actions are 0.
  - $q(1,5) = q(1,5) + \mathcal{R}_1^5 + 0.8 \max[q(5,1), q(5,4), q(5,5)] = 100$
- After reaching state 5, we finish one "episode".

$$Q = \begin{array}{c|c} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R = \begin{array}{c|c} & \text{Action} \\ \text{State} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

- For the second episode, assume we start at state 3.
- From $\mathcal{R}$, we have 3 possible actions: go to state 1, 2 or 4. Suppose we choose state 1.
- At state 1, we should update the Q table: $q(3,1) = q(3,1) + \mathcal{R}_3^1 + 0.8\max[q(1,5), q(1,3)] = 80$
- The next step should be the same as above.

$$Q = \begin{array}{c|c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \text{Action} \\ \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \end{array}$$

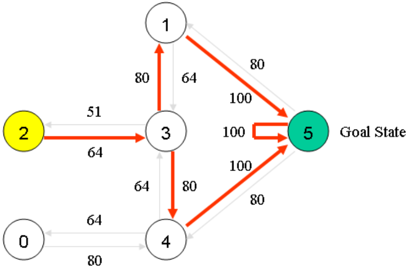
$$R = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix}$$

- After a few arounds of training, the Q table converged to the left.
- We normalize it by dividing the largest value.

$$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

- The final graph looks like this.
- The optimal path is just following the max value.

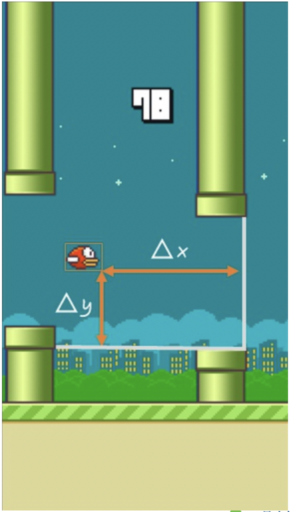


$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

Flappy Bird Example

- Let look at a more interesting example.
- Flappy Bird

- State $\mathcal{S}$
  - Obviously, we could choose each frame of game to be the state.
  - For simplicity concern, we choose $(\Delta x, \Delta y)$ to be the state.
  - Since we could not make the Q-table infinite, the $(\Delta x, \Delta y)$ are floored according to pixels.
- Action $\mathcal{A}$
  - In each frame, it could fly up or do nothing.
- Reward $\mathcal{R}$
  - For each frame, when the bird is alive, we give it reward 1. When the bird dies, we give it reward -1000.



- We try to train a Q-table, such as this:

State	Fly	Don't Fly
$(\Delta x_1, \Delta y_1)$	3	20
$(\Delta x_1, \Delta y_2)$	100	-20
$\dots$	$\dots$	$\dots$
$(\Delta x_m, \Delta y_{n-1})$	-100	1
$(\Delta x_m, \Delta y_n)$	3	-400

- The total number of states are $n \times m$, and each state has two actions.
- Theoretically, we could choose the maximum value each frame and survive infinite times.

- We could should the Q-learning algorithm more vigorously:

Algorithm 1: Q-learning

```
Initialization Q={};
while Q not converge do
    Initialize state s, start a new game;
    while s is not dead state do
        a = π(s); // Use ϵ-greedy.
        Use a in game, get new state s′ and reward R_s^a
        q(s, a) ← (1 − α)q(s, a) + α[R_s^a + γ max_{a′} q(s′, a′)]
        S ← S′
    end
end
```

- Let's try it !
- Flappy Bird

Thank You!