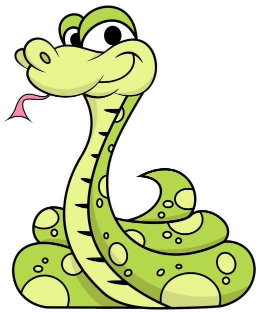
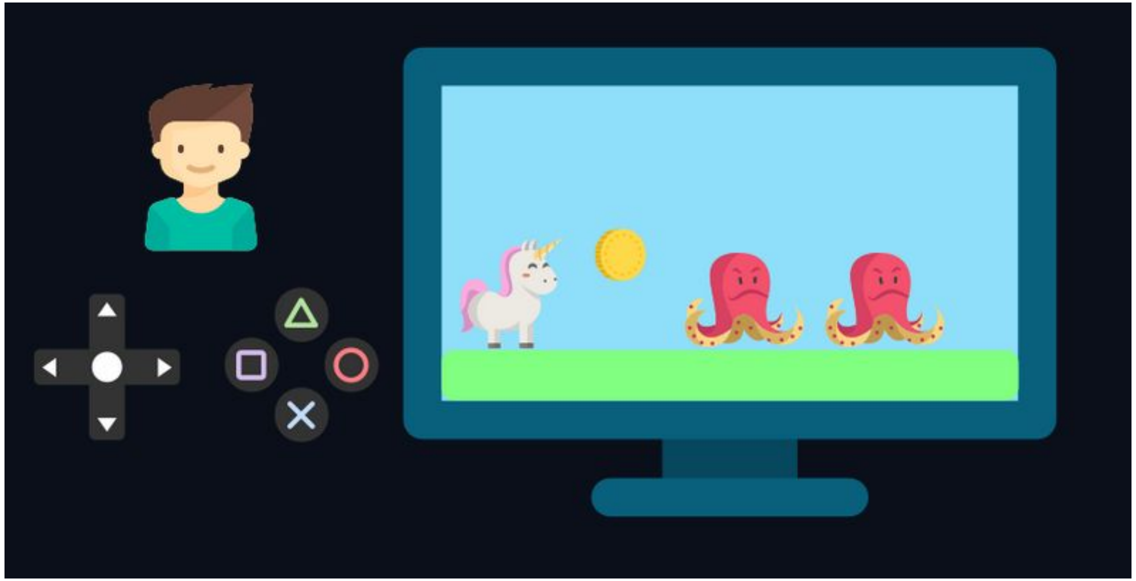
Learn2Slither: Intro to Reinforcement Learning

Stanislav Liashkov
42 Bangkok Student, Data Scientist



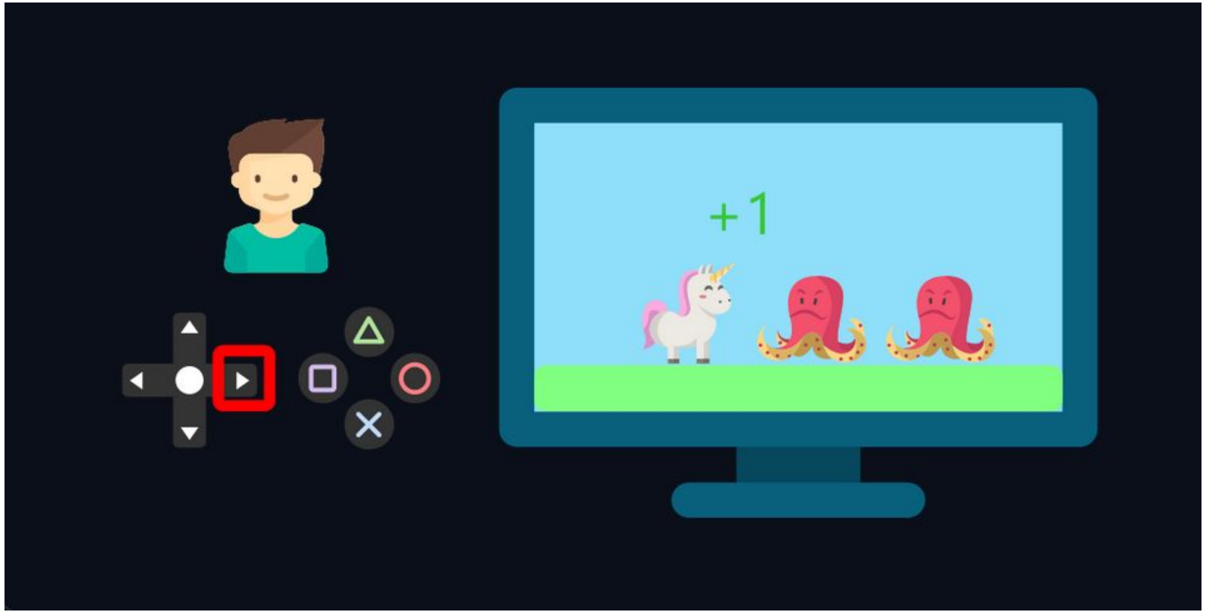
Example of Learning through Interaction

Imagine you have a little brother and you put him to play a video game he never played… He has some controllers and he can play by pressing buttons. But what buttons to press?



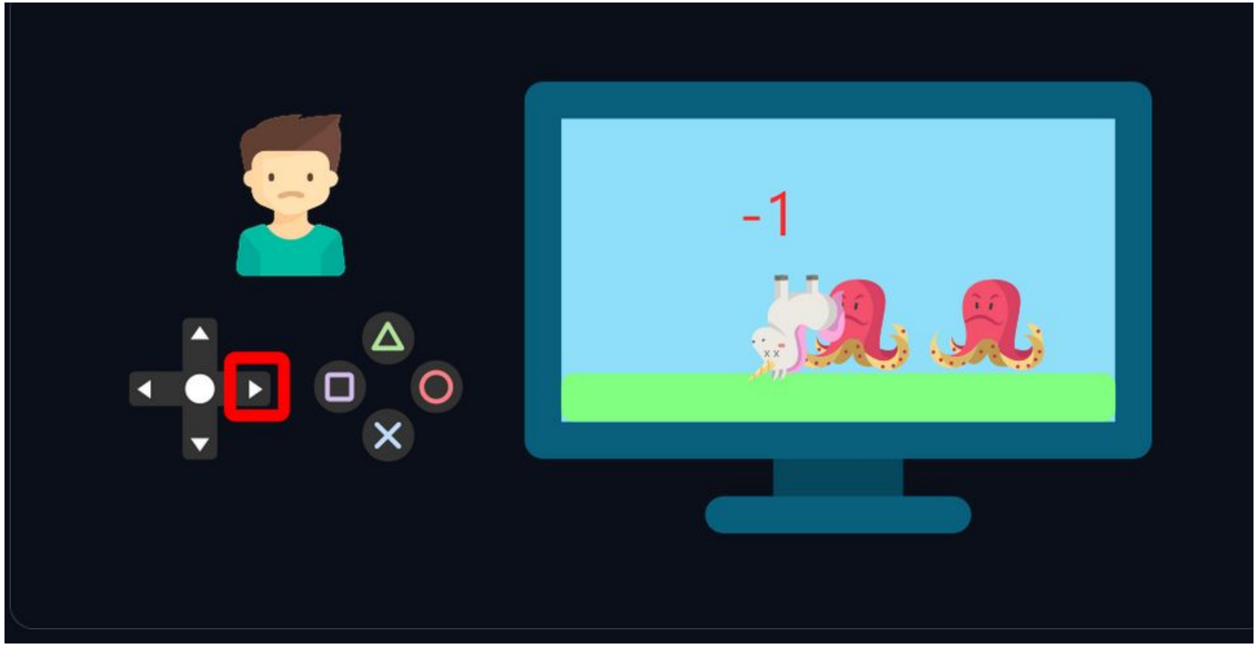
Example of Learning through Interaction

He tried to press **RIGHT** button and he **got a coin**! Well done! He is on the right track!



Example of Learning through Interaction

He tried to press the same button again and **hit the ENEMY!** He lost 1 reward point! This game is more complicated than he thought…



What is Reinforcement Learning?

Formal definition:

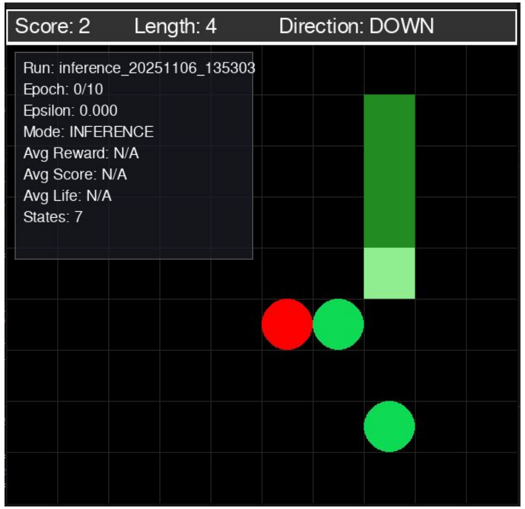
Reinforcement learning is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.

Idea: AI can learn through interaction with environment what actions to take to maximize cumulative expected reward.

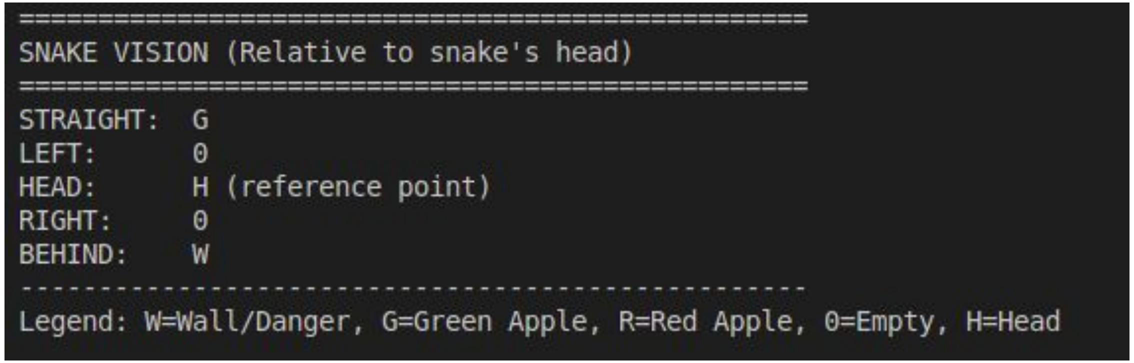
State (Observation)

State is information that agent gets from environment. State is used by agent to make decision what action to take.

In case of Snake Game example, state is what snake can see. (screen on the right)



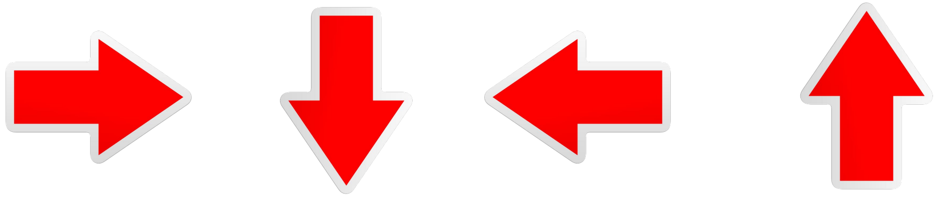
```
Score: 2        Length: 4        Direction: DOWN
Run: inference_20251106_135303
Epoch: 0/10
Epsilon: 0.000
Mode: INFERENCE
Avg Reward: N/A
Avg Score: N/A
Avg Life: N/A
States: 7
```

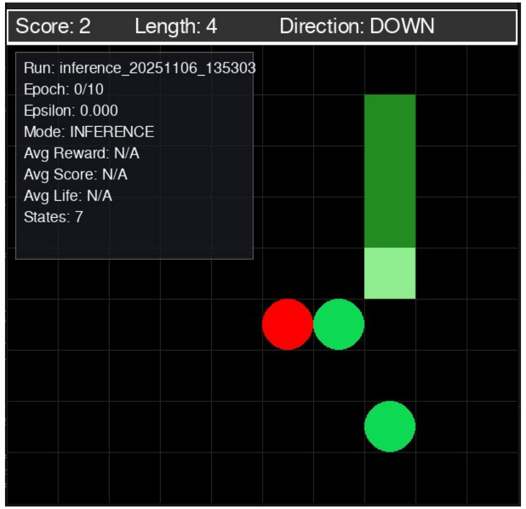


```
==================================================
SNAKE VISION (Relative to snake's head)
==================================================
STRAIGHT:   G
LEFT:       0
HEAD:       H (reference point)
RIGHT:      0
BEHIND:     W
--------------------------------------------------
Legend: W=Wall/Danger, G=Green Apple, R=Red Apple, 0=Empty, H=Head
```

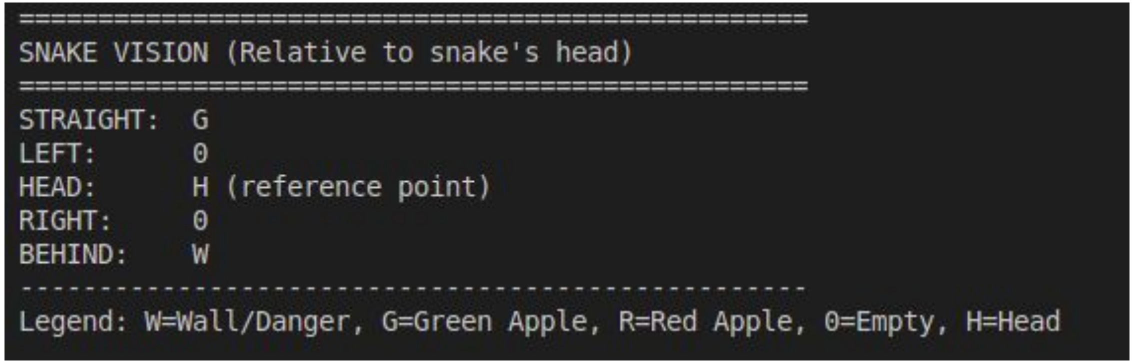
Action

For each **state**, there is a set of **actions** that agent can choose from.




Score: 2 Length: 4 Direction: DOWN

Run: inference_20251106_135303
Epoch: 0/10
Epsilon: 0.000
Mode: INFERENCE
Avg Reward: N/A
Avg Score: N/A
Avg Life: N/A
States: 7


```
============================================================
SNAKE VISION (Relative to snake's head)
============================================================
STRAIGHT:   G
LEFT:       0
HEAD:       H (reference point)
RIGHT:      0
BEHIND:     W
------------------------------------------------------------
Legend: W=Wall/Danger, G=Green Apple, R=Red Apple, 0=Empty, H=Head
```

Reward Function

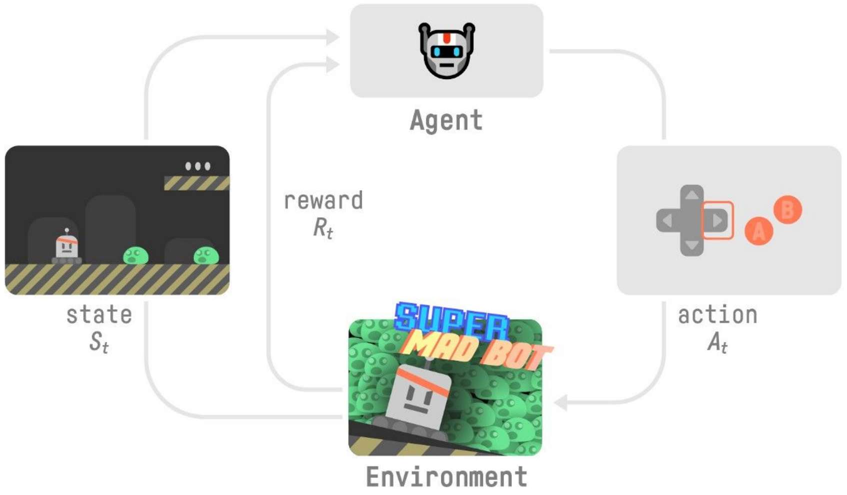
Reward function gives agent feedback on action he made in given step. If reward is positive, then agent did ‘good’ action if negative - bad action.

Agent will try to choose actions that returns maximal reward.

Reward(Current state, Action taken) = some value

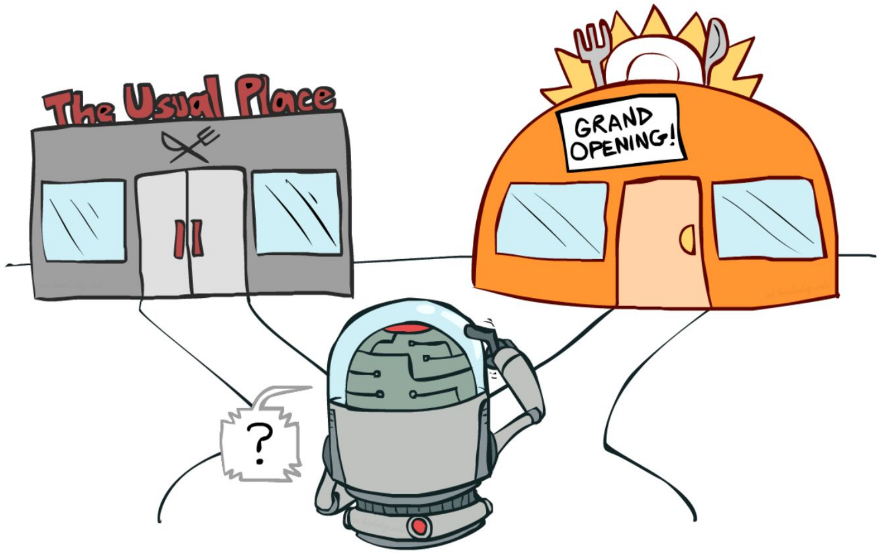
For snake: hitting a wall (game over) will have large negative reward and eating green apples will have small positive reward.

By designing reward func this way, we force agent to avoid walls and try to eat more apples to **maximize total return.**



Exploration vs Exploitation

- *Exploration* is exploring the environment by trying **random actions in order to find more information about the environment.**
- *Exploitation* is **exploiting known information to maximize the reward.**

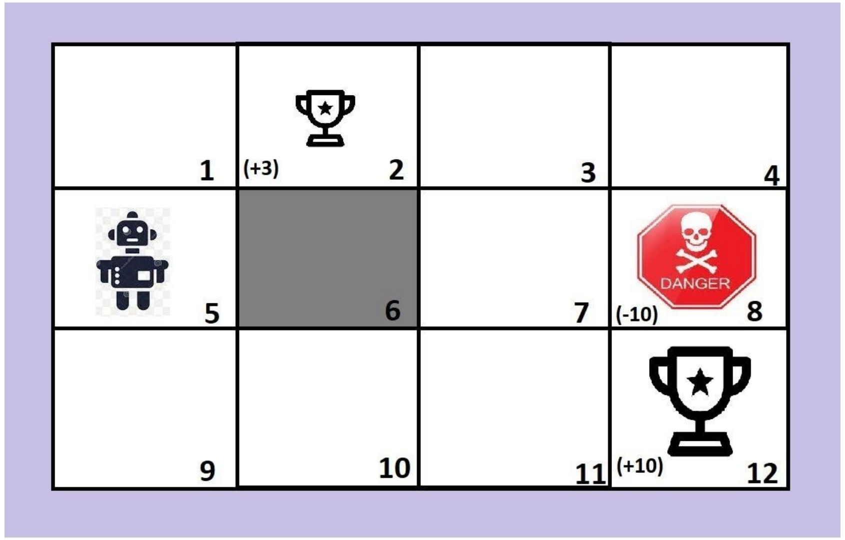


Exploration vs Exploitation

If agent does none or little exploration, it may get stuck in sub-optimal policy and exploit its knowledge, without learning new things.

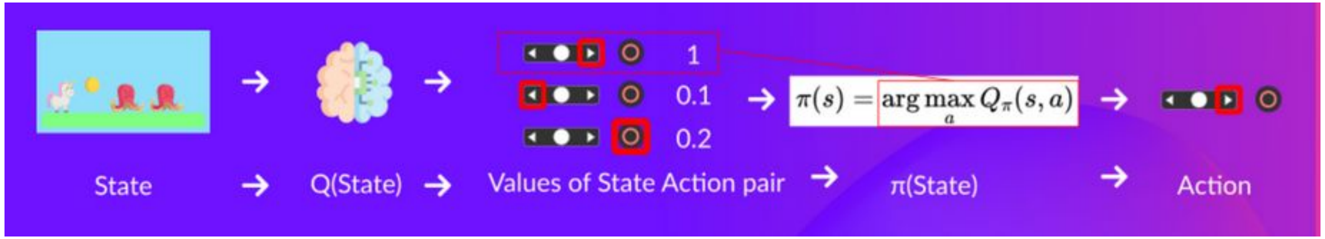
For example, (see pic) the agent will always try to reach block #2 with small reward and may never discover that block#12 has much more reward!

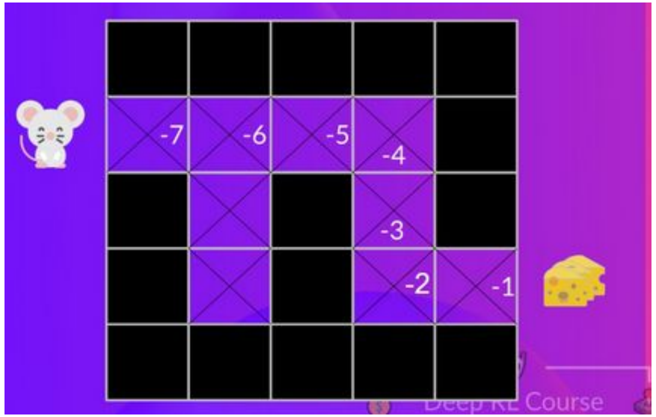
That’s why we need exploration.



Value-Based Methods

We train **action-value function** that for any pair of state + action will return **expected return**. Agent will use this value function to select action that maximizes value function.



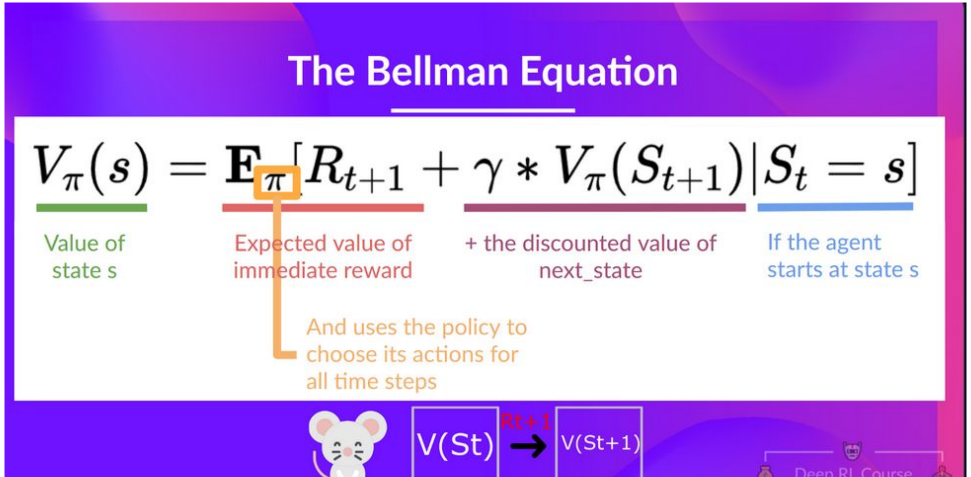


Bellman Equation

Value function of state is expected discounted return given starting state s and following policy P. (gamma is between 0 and 1)

$$\underbrace{v_\pi(s)}_{\text{Value function}} = \mathbb{E}_\pi\Big[\underbrace{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots}_{\text{Expected discounted return}} \mid \underbrace{S_t = s}_{\text{Starting at state s}}\Big]$$

Bellman Equation simplifies value estimation as follows: value equals **immediate reward of next state + discounted value of next state**



The Bellman Equation

$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{\mathbf{E}_{\boxed{\pi}}[R_{t+1}}_{\text{Expected value of immediate reward}} + \underbrace{\gamma * V_\pi(S_{t+1})}_{\text{+ the discounted value of next\_state}} | \underbrace{S_t = s]}_{\text{If the agent starts at state s}}$$

And uses the policy to choose its actions for all time steps

Q-table

Q(State, Action) is a ‘quality’ function that returns expected cumulative reward if agent starts at given state and makes given action.

This function is essentially **a table or a cheat sheet** that helps agent make decisions.

straight	left	right	behind	action	Q-value
🍏	none	none	🧱	⬆	+22.34
🧱	🍏	🍎	🍏	➡	+4.32
none	🧱	none	🍎	⬅	-912.78
…	….	…	…	…	…

Q-learning Algorithm

1. Initialize Q-table at zeros (or randomly)
2. Given current state, select random action at probability **ε**, and select action that maximizes Q-value with probability **1-ε**
3. Update Q-table
4. Repeat for n steps

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}}_{\text{TD Error}}]$$

Common Applications of RL

AI players

Robotics

Online Advertising

Self-driving cars