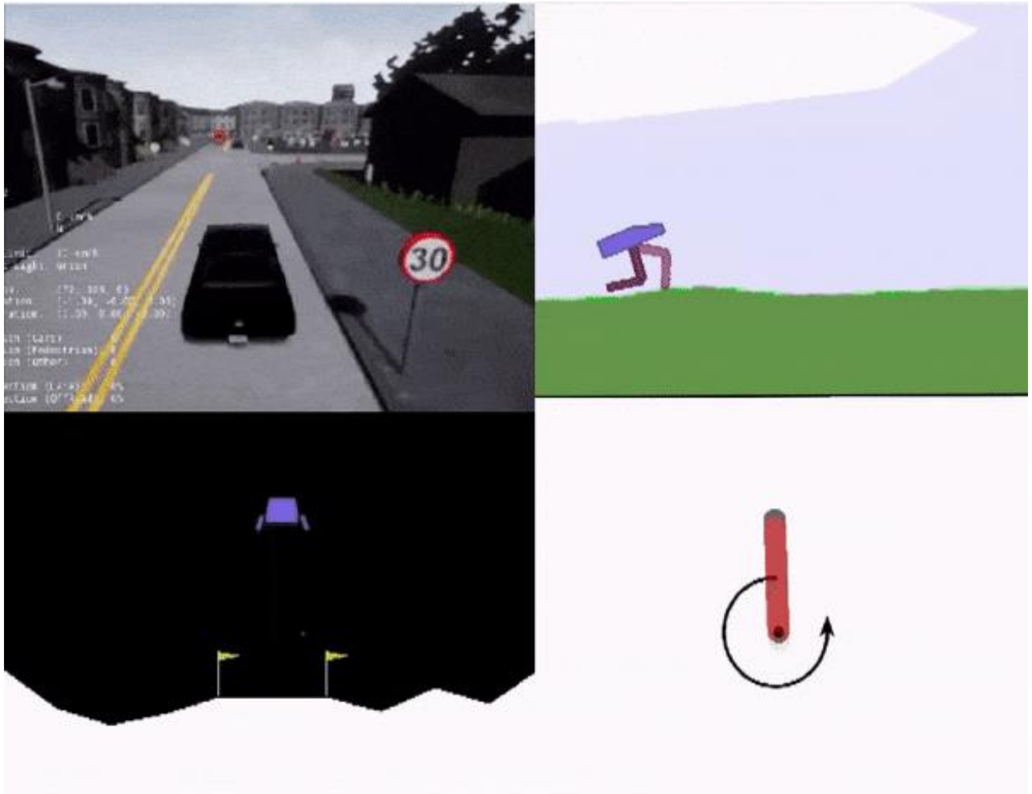
RL-TRAINED TAXI TUTORIAL
USING Q-LEARNING

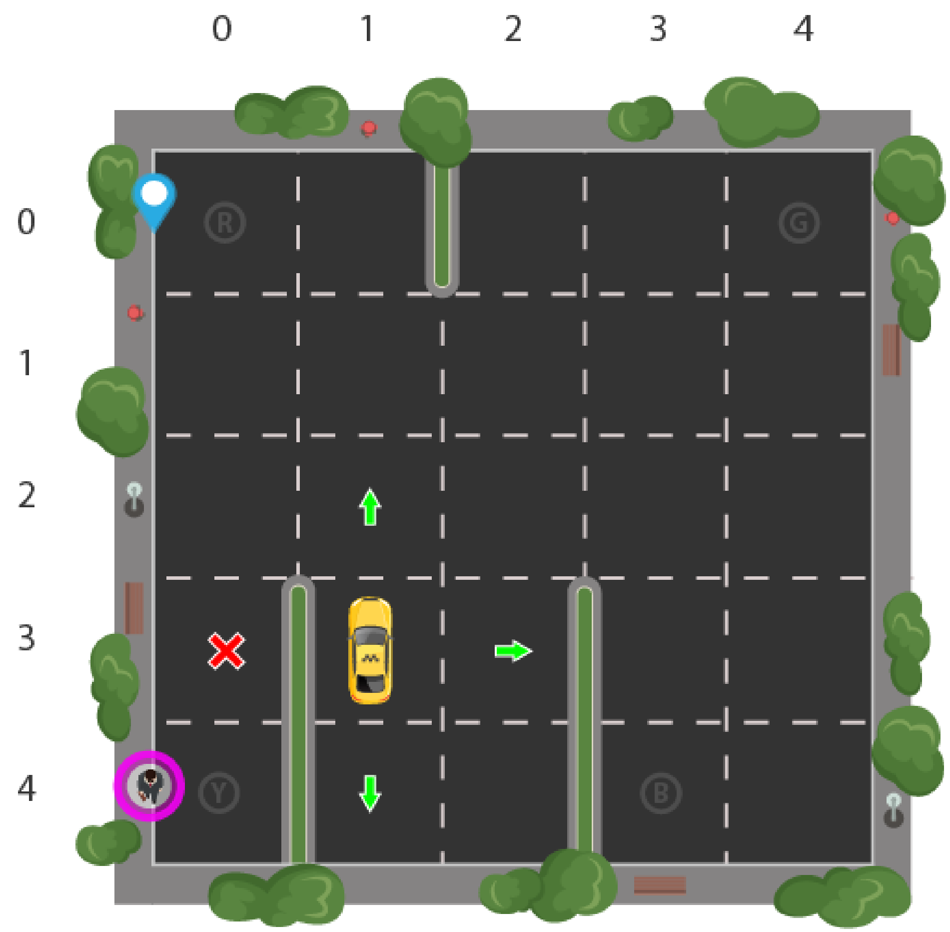
Introducing OpenAI Gym

- OpenAI gym is a standard API for developing and testing learning agents; **suited for reinforcement learning use cases**
- This **python library** gives us a huge number of test environments to work on our RL agent's algorithms with shared interfaces for writing general algorithms and testing them



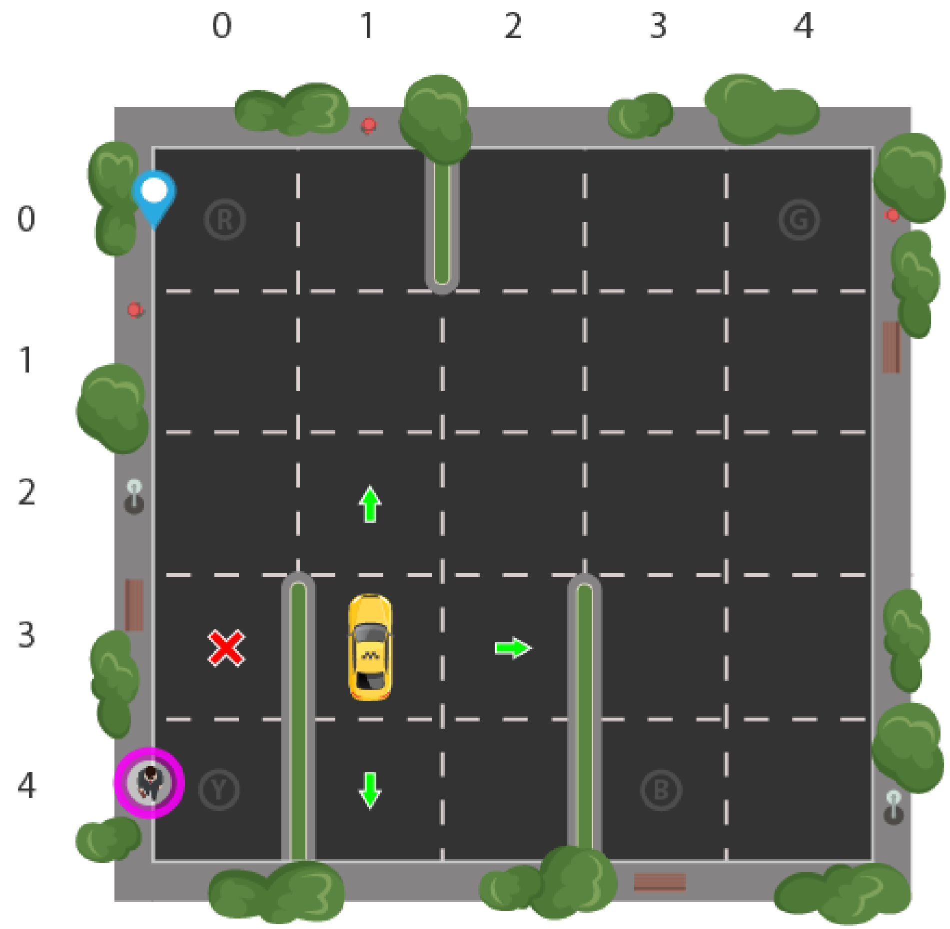
Introducing Taxi Environment

- Taxi is **one of many environments available on OpenAI Gym**. These environments are used to develop and benchmark reinforcement learning algorithms
- **The goal of Taxi is to pick-up passengers and drop them off at the destination in the least amount of moves**



Introducing Taxi Environment

- 4 designated locations in the grid world indicated by **R(ed), G(reen), Y(ellow), and B(lue)**
- When the episode starts, **the taxi starts off at a random square and the passenger is at a random location**
- The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the 4 locations), and then drops off the passenger
- **Once the passenger is dropped off, the episode ends**

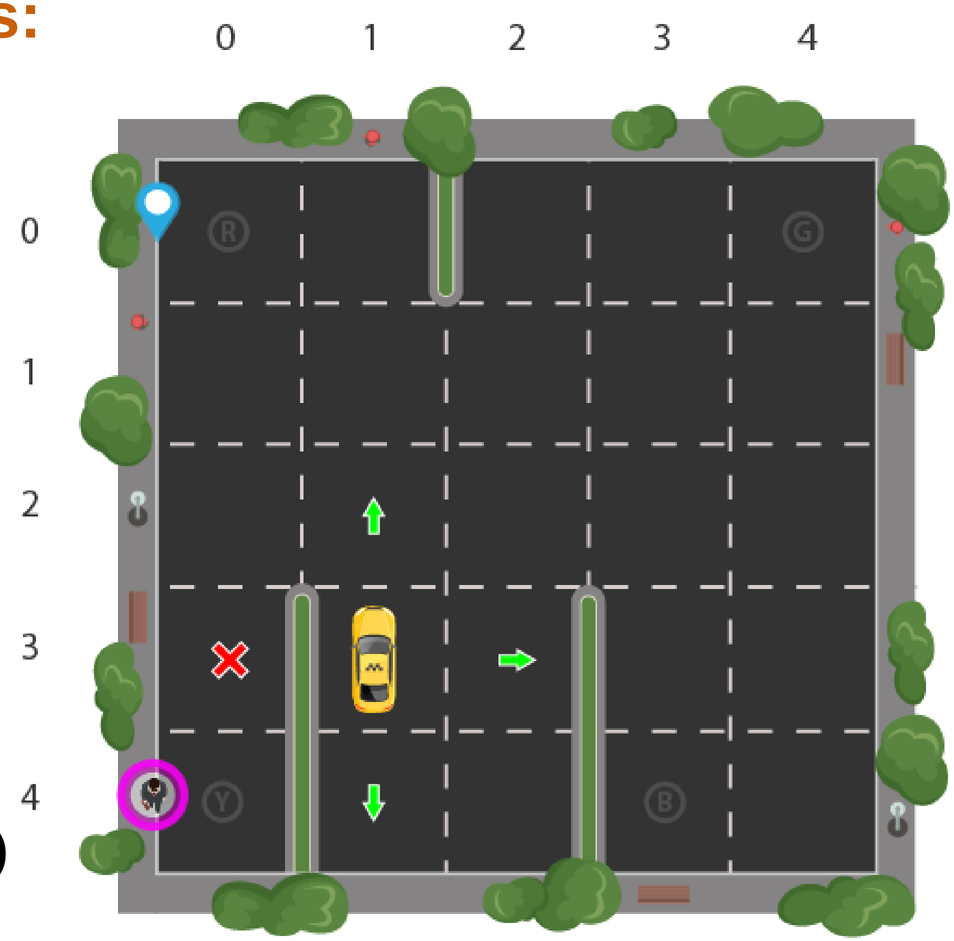


Introducing Taxi Environment

Various Possible Actions:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

Each state space is represented by the tuple: (*taxi_row, taxi_col, passenger_location, destination*)



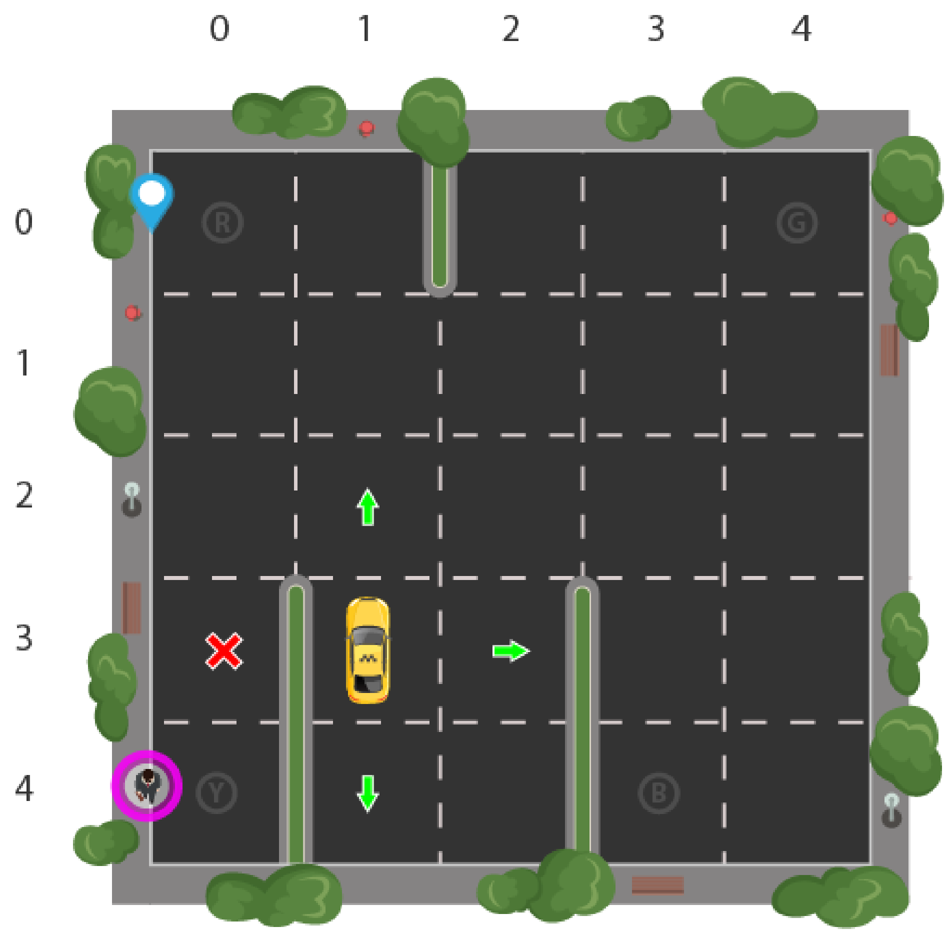
Introducing Taxi Environment

Passenger locations:

- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)
- 4: in taxi

Destinations:

- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)



Taxi Reward System

Action	State	Reward / Penalty
Any successful movements (south/north/west/east)	N.A.	-1
Pickup or drop-off passenger wrongly	Wrong pickup/drop-off point (R/G/Y/B)	-10
Drop off passenger correctly	Correct drop-off point (R/G/Y/B)	+20

Introducing Q Learning

- ‘Q’ = Quality = how valuable a given action is in gaining future reward
- Q-values = State-Action values
- Q-learning is a model-free reinforcement learning algorithm that **seeks to find the best action to take given the current state, i.e. to seek to learn a policy that maximizes the total reward**
- **Off-policy** – the Q-learning function **learns from actions that are outside the current policy (i.e. greedy policy), like taking random actions**; a policy isn’t needed
- In contrast, **On-policy** **learns from actions that are based on the current policy**
- Can handle problems with stochastic transitions and rewards without requiring adaptations

Introducing Q Learning

Tic-Tac-Toe Example:

- No. of states = 765 because that is the total number of possible valid board states in Tic-Tac-Toe
- Q-Table is a type of policy that assigns each state-action pair a Q-Value individually using a table of values, rather than using some sort of function that takes the state as input
- **When the policy is used to pick an action at a given state, the action with the highest Q-Value in that state is picked**

1	2	3
4	5	6
7	8	9

		States				
		1	2	3	...	765
Actions	1	Q(1, 1)	Q(2, 1)	Q(3, 1)	...	Q(765, 1)
	2	Q(1, 2)	Q(2, 2)	Q(3, 2)	...	Q(765, 2)
	3	Q(1, 3)	Q(2, 3)	Q(3, 3)	...	Q(765, 3)
	4	Q(1, 4)	Q(2, 4)	Q(3, 4)	...	Q(765, 4)
	5	Q(1, 5)	Q(2, 5)	Q(3, 5)	...	Q(765, 5)
	6	Q(1, 6)	Q(2, 6)	Q(3, 6)	...	Q(765, 6)
	7	Q(1, 7)	Q(2, 7)	Q(3, 7)	...	Q(765, 7)
	8	Q(1, 8)	Q(2, 8)	Q(3, 8)	...	Q(765, 8)
	9	Q(1, 9)	Q(2, 9)	Q(3, 9)	...	Q(765, 9)

Q-Table

Introducing Q Learning

Q-learning Algorithm:

Q-value
(for a state (S) and action(A))

Reward

Maximum expected
future reward

$$\boxed{Q(S_t, A_t)} \leftarrow \boxed{Q(S_t, A_t)} + \boxed{\alpha}\left[\boxed{R_{t+1}} + \boxed{\gamma}\boxed{\max_a Q(S_{t+1}, a)} - \boxed{Q(S_t, A_t)}\right]$$

Learning rate Discount factor

New Q Value Old Q Value Old Q Value

Note that $Q^{new}(s_t, a_t)$ is the sum of three factors:

- $(1-\alpha)Q(s_t, a_t)$: the current value weighted by the learning rate. Values of the learning rate near to 1 make the changes in Q more rapid.
- $\alpha\, r_t$: the reward $r_t = r(s_t, a_t)$ to obtain if action a_t is taken when in state s_t (weighted by learning rate)
- $\alpha\gamma \max_a Q(s_{t+1}, a)$: the maximum reward that can be obtained from state s_{t+1}(weighted by learning rate and discount factor)

Introducing Q Learning

Q-learning Algorithm:

Temporal Difference

Q-value
(for a state (S) and action(A))

Reward

Maximum expected
future reward

$$\underbrace{Q(S_t, A_t)}_{\text{Q-value}} \leftarrow Q(S_t, A_t) + \underbrace{\alpha}_{\text{Learning rate}} \overbrace{\left[\underbrace{\overbrace{R_{t+1}}^{\text{Reward}} + \underbrace{\gamma}_{\text{Discount factor}} \overbrace{\max_a Q(S_{t+1}, a)}^{\text{Maximum expected future reward}}}_{\text{New value (temporal difference target)}} - Q(S_t, A_t) \right]}^{\text{Temporal Difference}}$$

Learning rate

Discount factor

New value (temporal difference target)

Familiar? Refer to the Bellman Equation

This algorithm will help our agent **update the current Q-value (*Q*(*St*,*At*)) with its observations after taking an action**, i.e. increase Q if it encountered a positive reward, or decrease Q if it encountered a negative one

Explore-Exploit Strategy

1. Exploit – The agent selects the action based on the max value of the state-actions in the Q-Table; **use the information we have available to us to make a decision**

2. Explore – Act randomly. **Instead of selecting actions based on the max future reward in the Q-Table, we select an action at random**; allows the agent to explore and discover new states

Balance exploration/exploitation by setting epsilon (ε) value [from 0 to 1]; higher ε-value means more exploration and less exploitation, and vice versa

Codes (Editable Parts)

Training; Section 3

```
# hyperparameters to tune (***Make changes here if you want***)
learning_rate = 0.9
discount_rate = 0.8
epsilon = 1.0
decay_rate= 0.005

# training variables (***Make changes here if you want***)
num_episodes = 2000
max_steps = 99 # per episode
```

Visualization; Section 4

```
# ***Make changes here if you want; you can increase the number of episodes***
episodes_to_preview = 10
```

Mini Task
(Do not need to submit)

- **Go to Section 3 and try various sets of hyper parameters when training the model**; the objective is to **maximize the accumulated rewards, denoted by 'Score' when testing in Section 4**
- You may try to **increase the number of training episodes** if you feel that it will help in the model performance
- You may try to test the optimal model with **a higher number of testing episodes** to check its performance

```
TRAINED AGENT
+++++EPISODE 10+++++
Step 16
Successful Dropoffs: 10 out of 10 episodes
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Score: 5
```

THANK YOU

Email: nicholas.ho@nus.edu.sg