

Welcome Everyone

AI Mastery Course



Modul Domain AI

Reinforcement Learning

Bagian

Introduction to Reinforcement
Learning

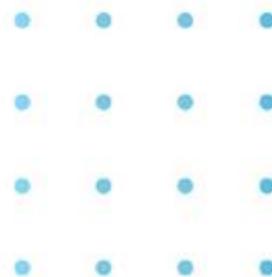




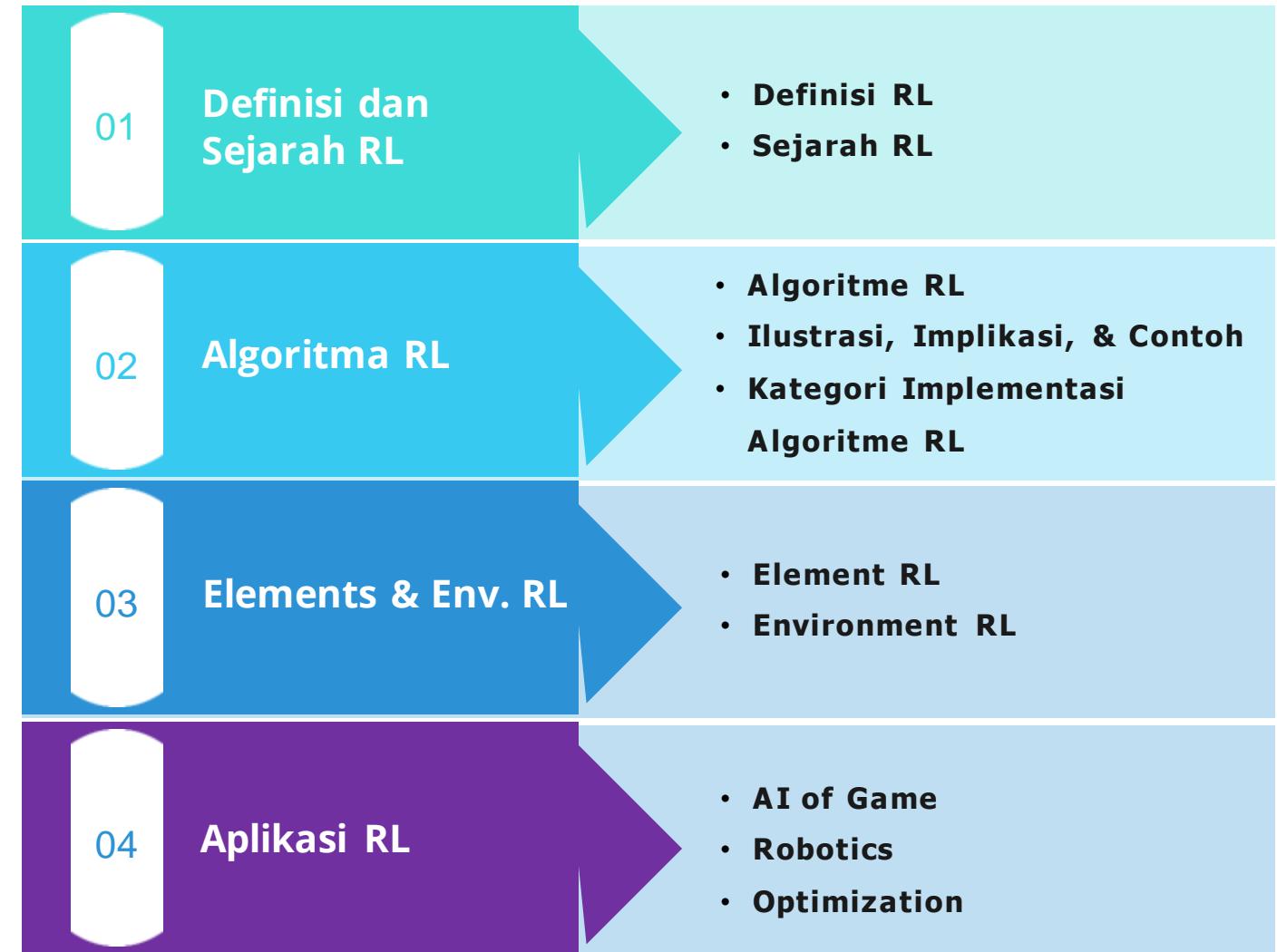
Learning Objectives

Mengetahui hal-hal berikut:

- Sejarah Reinforcement Learning (RL)
- Algoritma RL
- Elements & Environment dari RL
- Aplikasi dari RL



Agenda

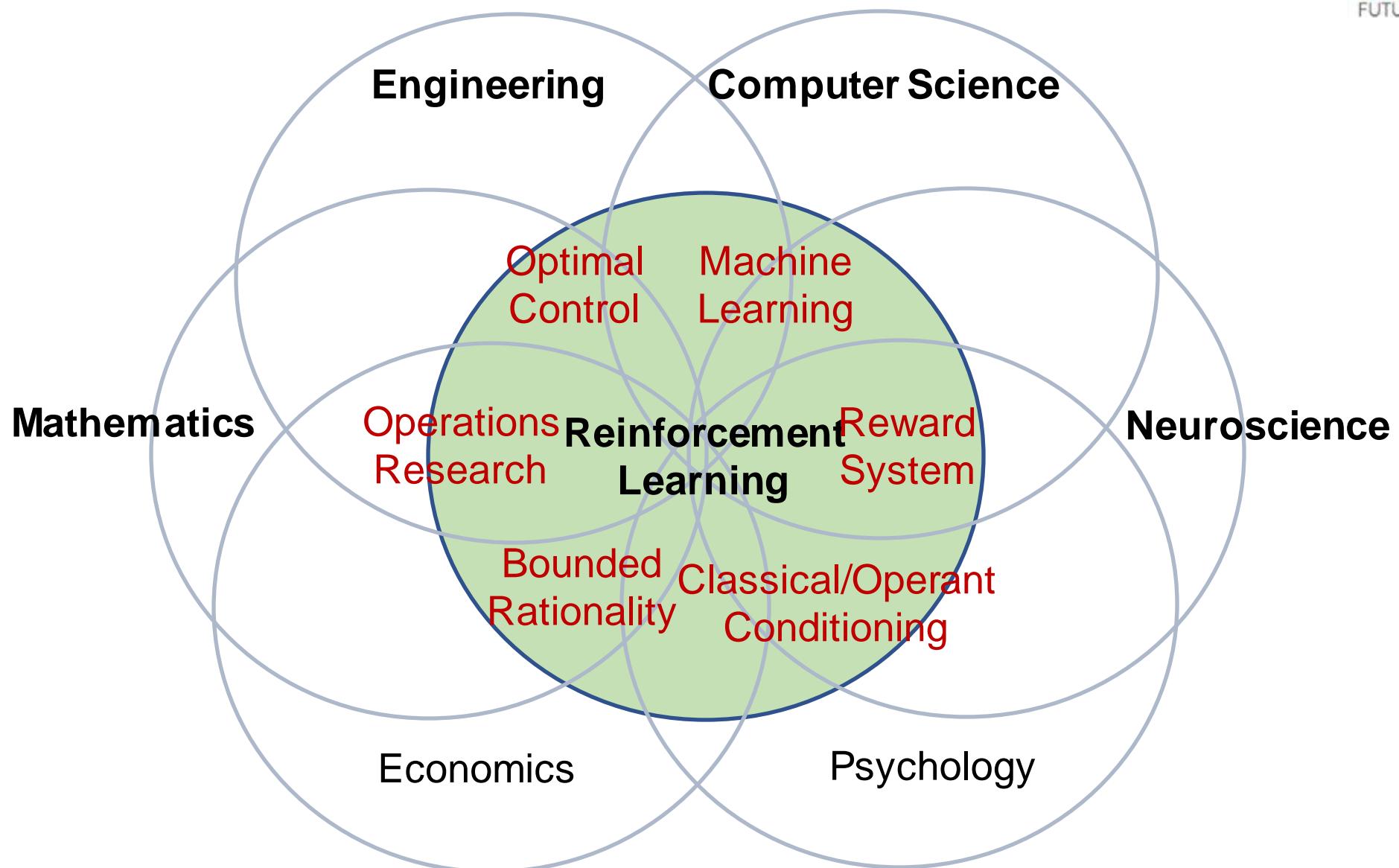




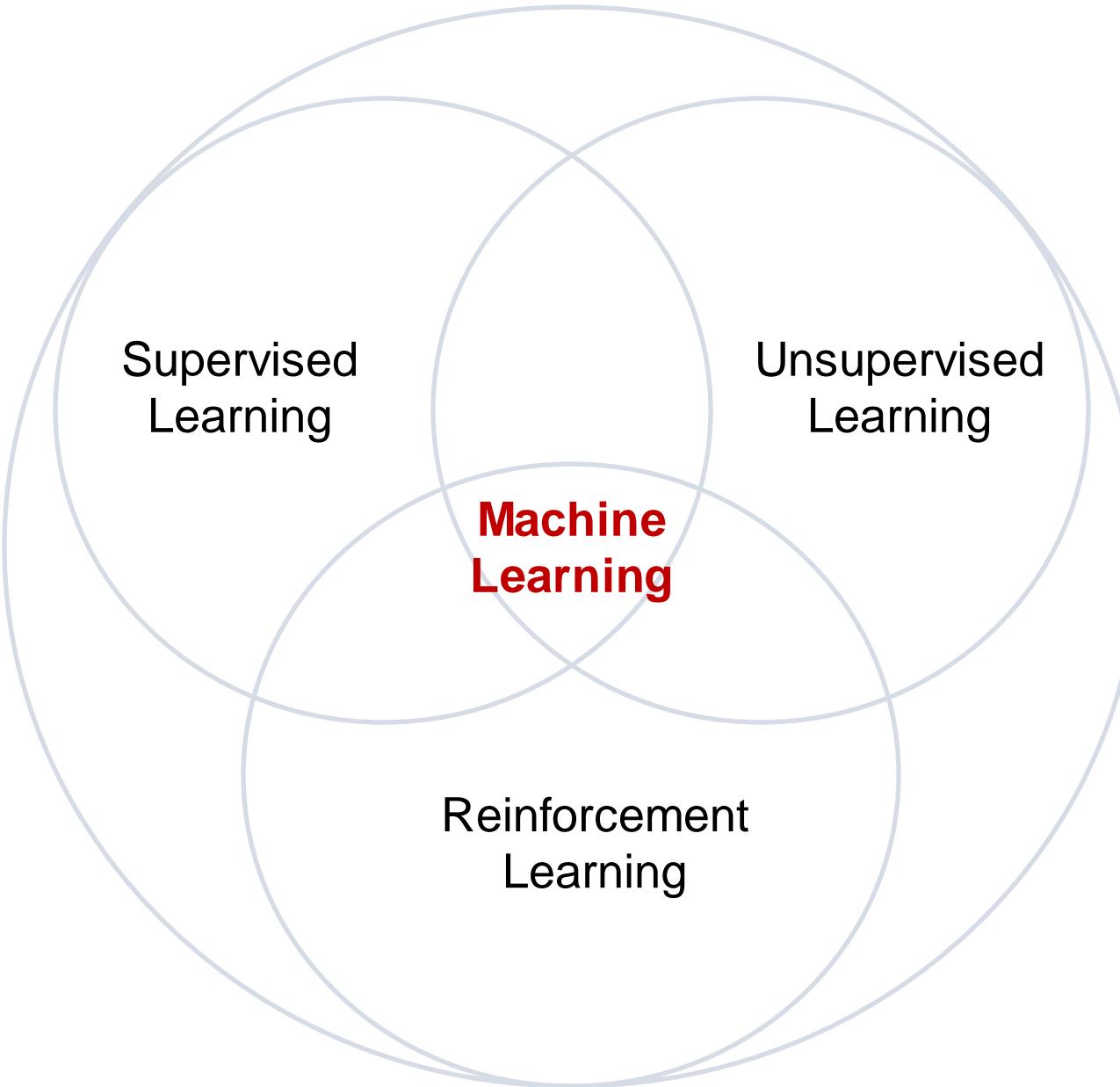
01

Definisi & Sejarah RL

Reinforcement Learning dalam Berbagai Bidang



Cabang Machine Learning



Stanford Marshmallow Test

- Tahun 1972
- Lebih lanjut lihat di:
https://en.wikipedia.org/wiki/Stanford_marshmallow_experiment



Agent, Reward, dan Action?

- **Agent:** Siapakah Agent dalam eksperimen marshmallow?
- **Reward:** Apakah Karakteristik *Reward* yang bisa kamu ketahui dari contoh di atas?
- **Action:** Tindakan apa saja yang tersedia pada Eksperimen Marshmallow?



Karakteristik Reward dan Action

- **Reward:**

Sistem *Reward* dalam sebuah Lingkungan mungkin tidak langsung diberikan. (*delayed reward*).

Reward bisa dilihat sebagai umpan balik terhadap *Agent*, karena beberapa *reward* mengalami penundaan maka umpan balik seringkali menjadi tertunda.

Ada yang namanya “*Delayed Gratification*” dan seringkali memberikan total *reward* yang lebih besar.

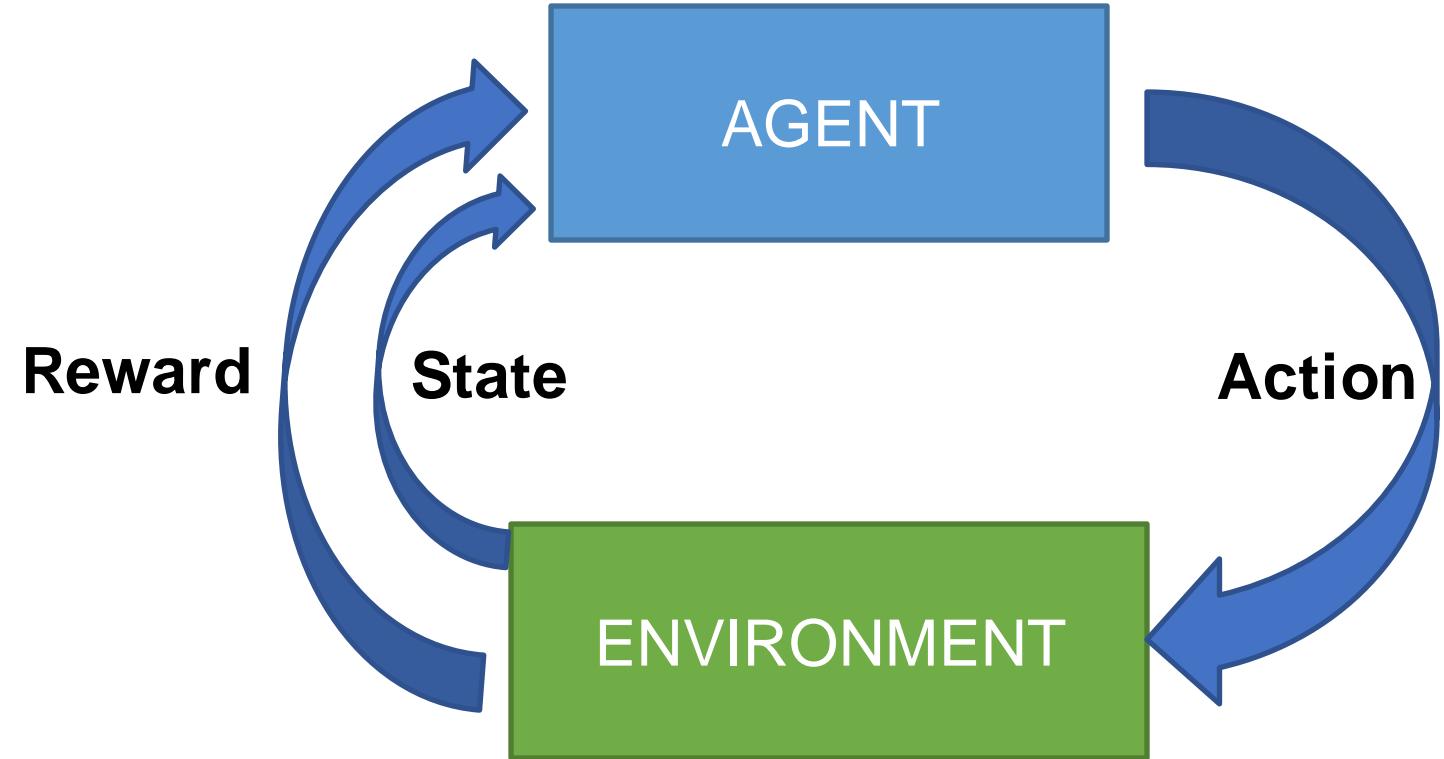
- **Action:**

Pada contoh di atas Semua Tindakan yang mungkin telah diketahui.



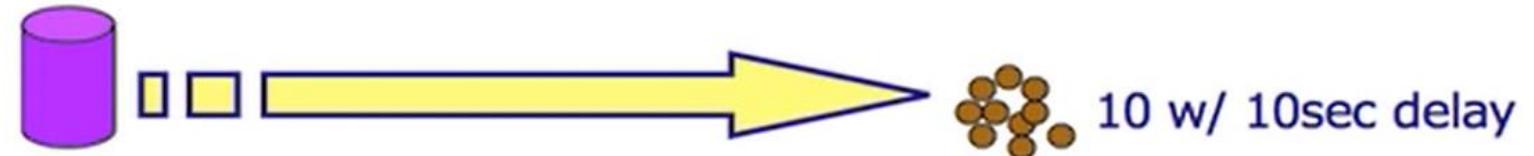
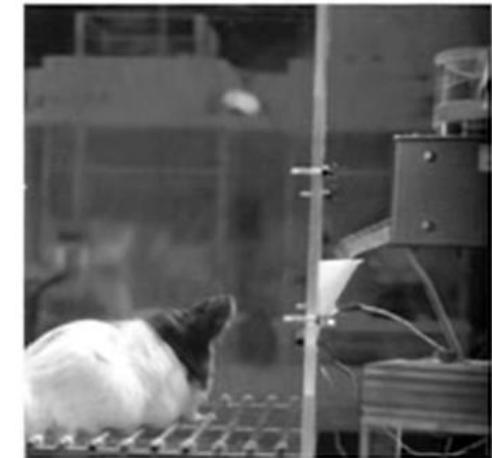
RL Secara Formal?

- Reward (R_t)
 - -
 - -
 - -
- State (S_t)
 - -
 - -
- Action (A_t)
 - -
 - -
 - -
 - -
 - -



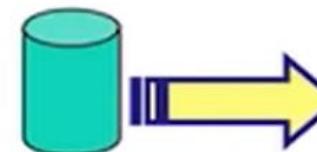
Trial & Error Systems 1

- Ainslie & Herrnstein 1974



Trial & Error Systems 2

- Ainslie & Herrnstein 1974



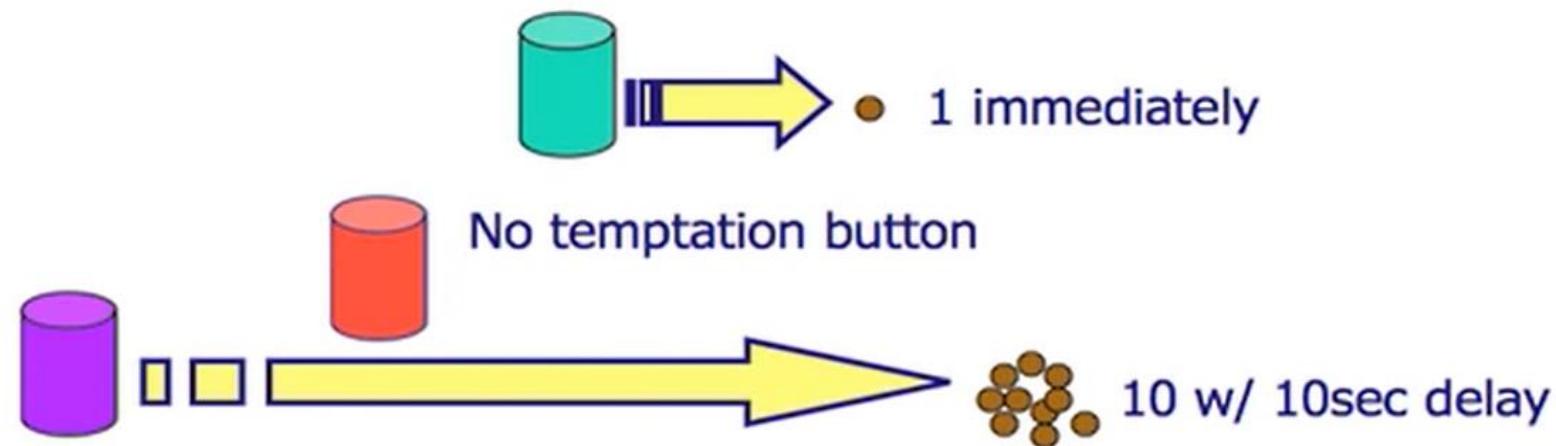
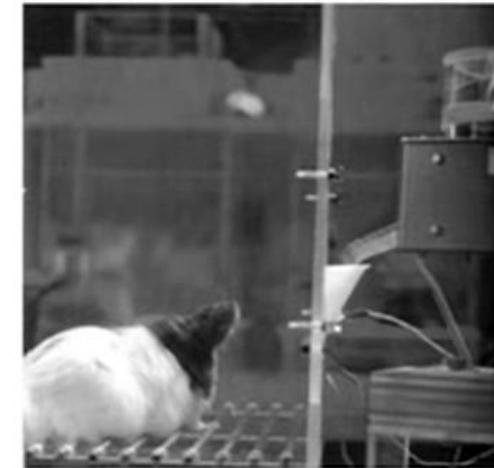
• 1 immediately



10 w/ 10sec delay

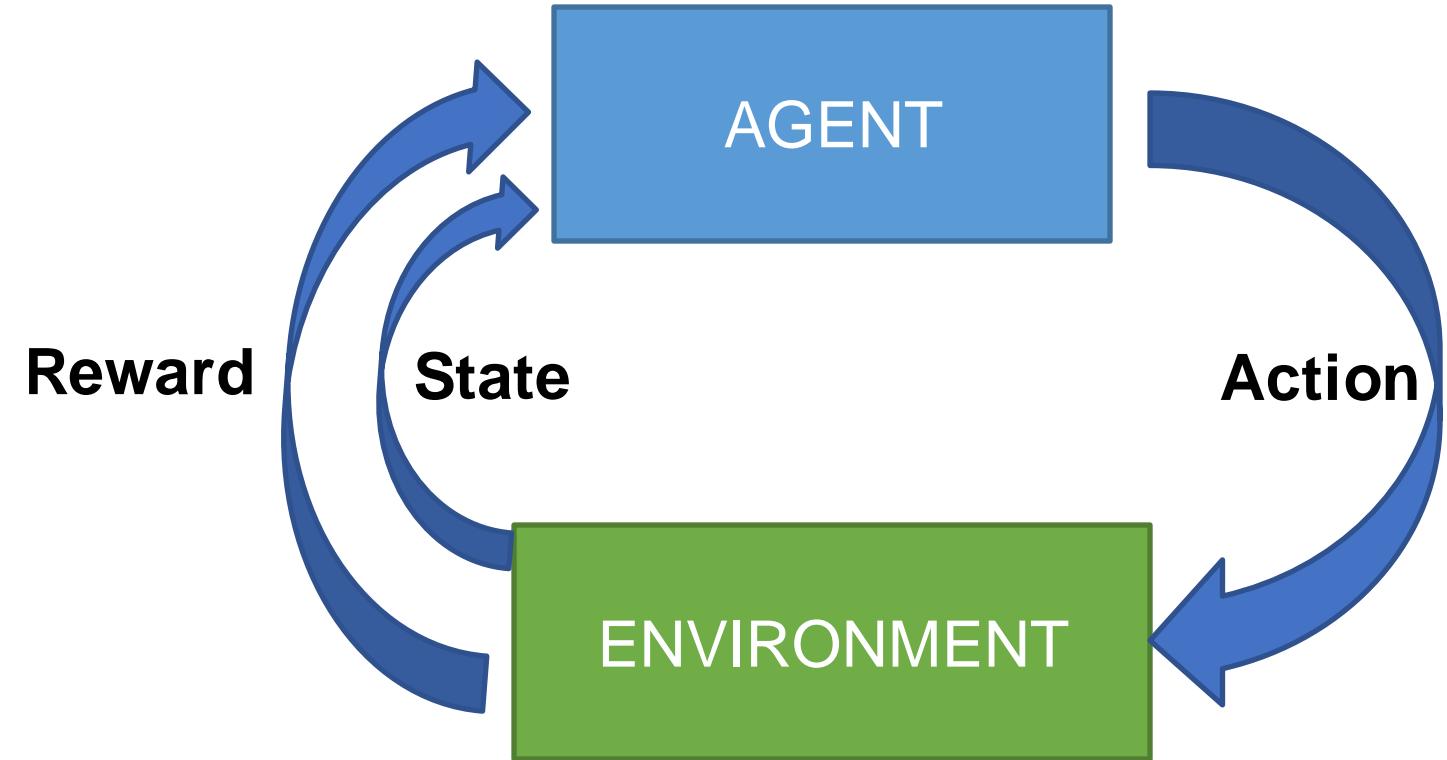
Trial & Error Systems 3

- Ainslie & Herrnstein 1974



RL Secara Formal?

- Reward (R_t)
 - -
 - -
 - -
- State (S_t)
 - -
 - -
 - -
- Action (A_t)
 - -
 - -
 - -



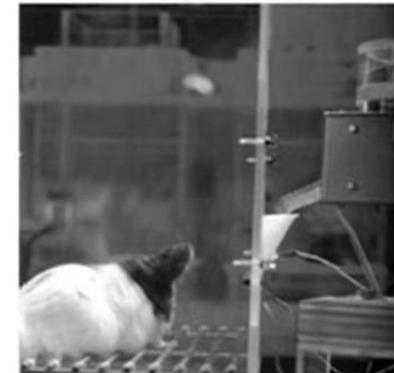
Environment (Lingkungan)?

- Apa yang membedakan lingkungan antara percobaan Marshmallow dan Merpati, Tikus?



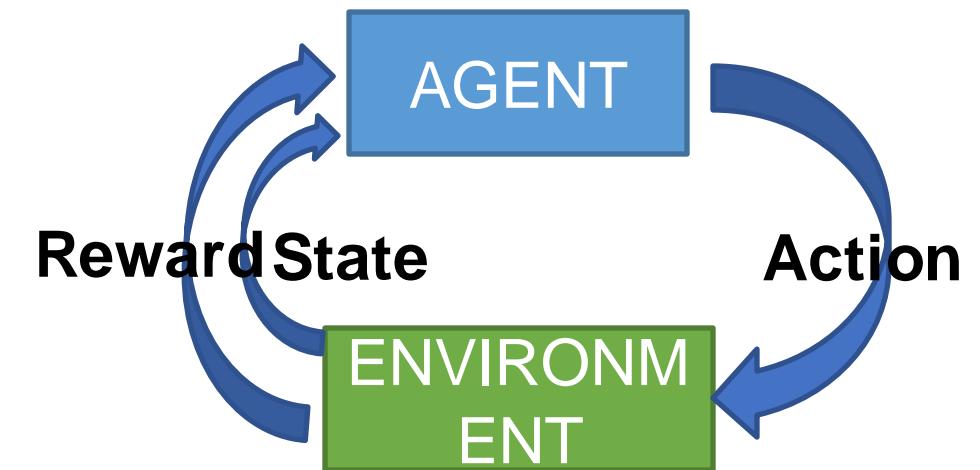
Environment (Lingkungan)

- Aturan Reward dalam suatu Lingkungan Kadang dideskripsikan dengan jelas.
- Ada Env yang perlu dipelajari karena tidak terdapat aturan yang jelas. (Trial and Error / Eksplorasi).
- Dalam Dunia nyata, Improvisasi masih bisa terjadi meski sudah ada aturan.



Karakteristik dari RL

- Tidak ada “kebenaran” pada proses pembelajaran (*training*) yang ada hanya hadiah (atau hukuman)
- Data yang diperoleh (*State* dan *Reward*) tergantung dari *Action* yang diberikan oleh *Agent*.
- Variabel waktu dan urut-urutan *state* yang dipilih oleh *Agent* melalui *action* memiliki pengaruh yang penting.



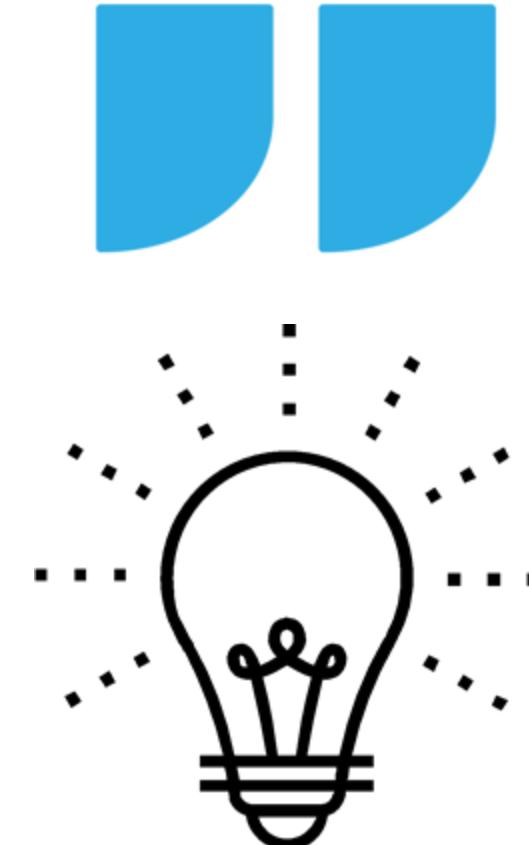
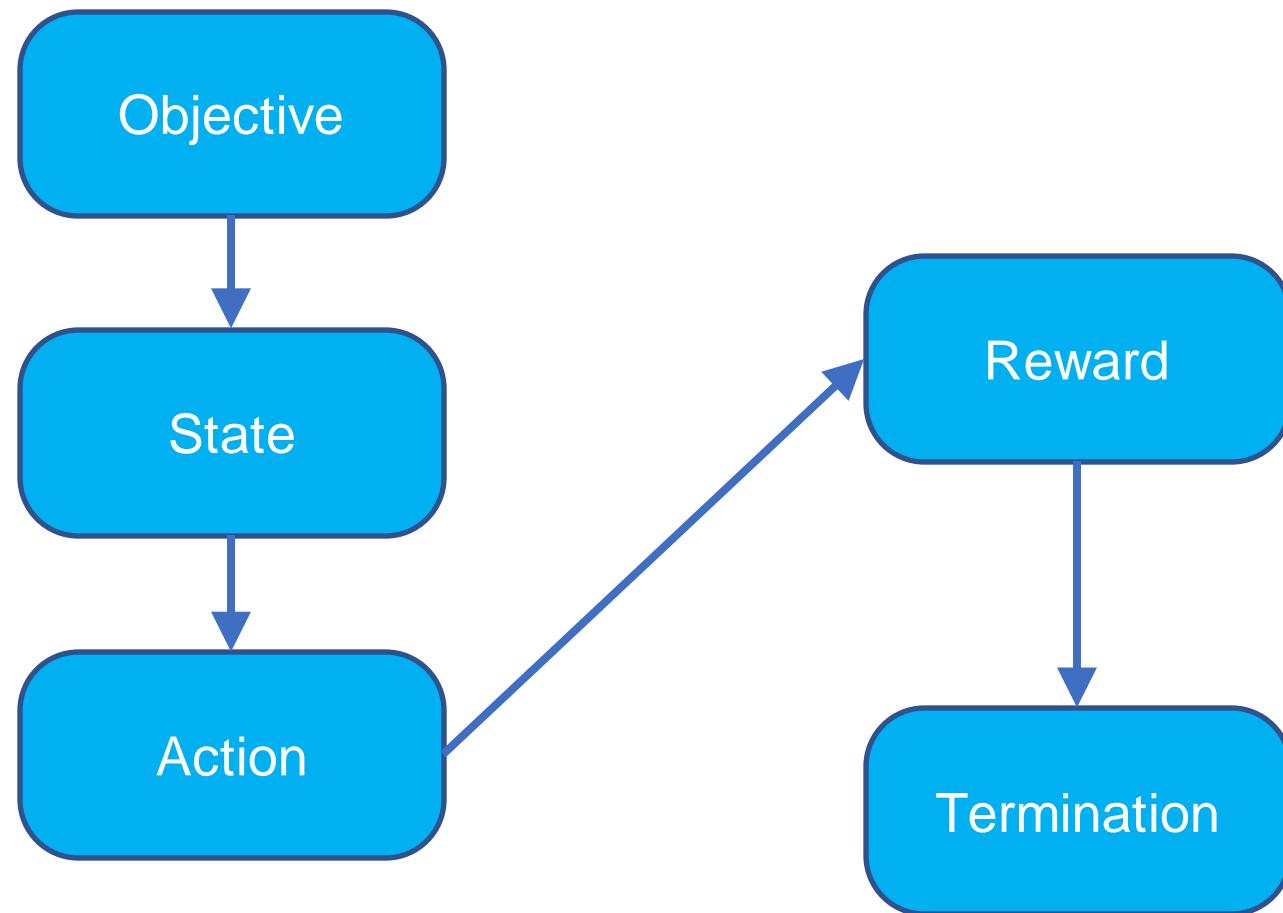


02

Algoritma RL

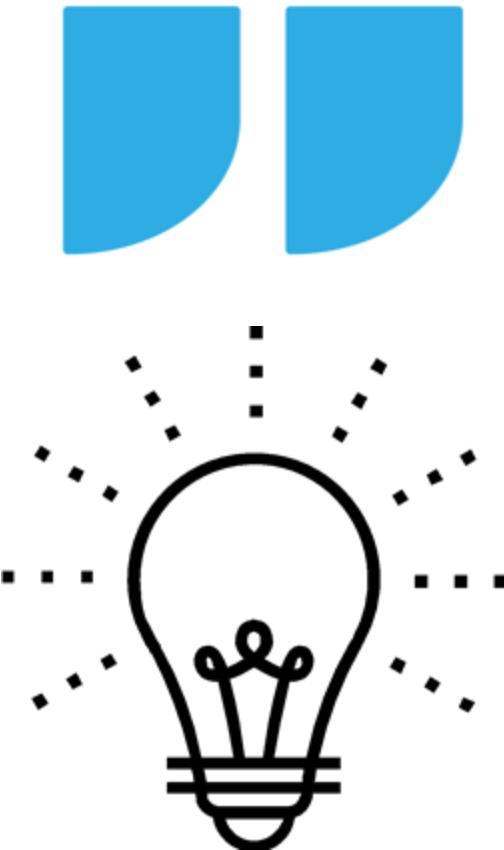
- Algoritma RL
- Ilustrasi, Implikasi, & Contoh
- Kategori Implementasi Algoritma RL

Algoritme RL



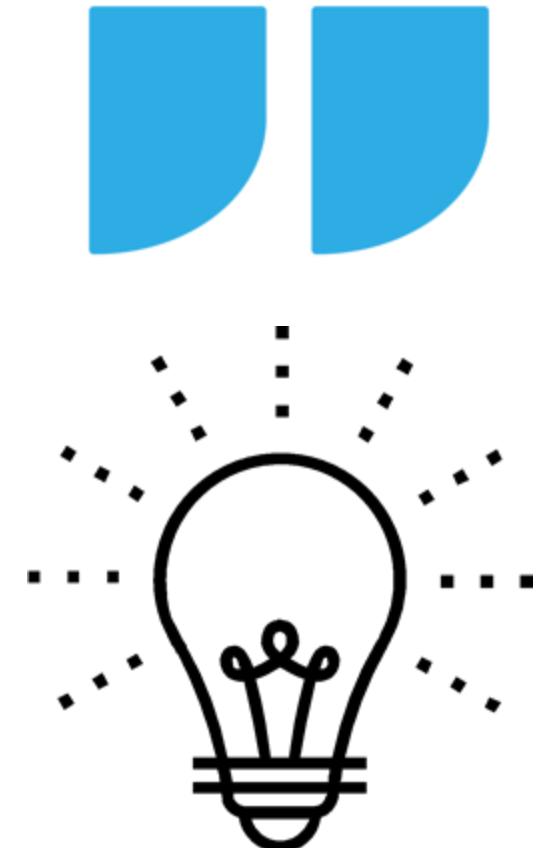
Algoritme RL (cont'd)

- Algoritme RL menghendaki pertukaran *state-action-reward* (s_t, a_t, r_t) antara *agent* dan *environment*. Proses ini dinamakan *sequential decision-making process*, dan (s_t, a_t, r_t) dinamakan *experience*.
- RL menghitung jumlah dari *reward* yang diterima oleh *agent*. Tujuan (objective) dari *agent* adalah memaksimalkan total *reward*.
- RL belajar (**learn**) dari interaksi *agent* dengan *environment* menggunakan proses *trial & error* dan *reward* yang diperoleh *agent*, untuk memperkuat (**reinforce**) aksi positif.



Algoritme RL (cont'd)

- Agent berinteraksi terhadap environment dengan melakukan sebuah aksi/tindakan dari satu kondisi ke kondisi yang lain.
- Agent akan menerima *reward* berdasarkan aksinya.
- Berdasarkan *reward* tersebut, *agent* akan memahami apakah aksinya baik atau buruk.
- Jika aksinya baik, agent akan menerima *reward* positif, sehingga agent tersebut akan cenderung lebih memilih melakukan aksi yang serupa dengan aksi tersebut (*exploitation*), atau melakukan aksi lain yang dapat menghasilkan *reward* positif.



Ilustrasi Reward pada Pergerakan Agent RL

Robot
Menjauhi
Bebatuan



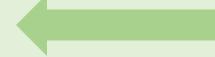
+ 10 Poin



Robot
Mendekati
Bebatuan



- 5 Poin



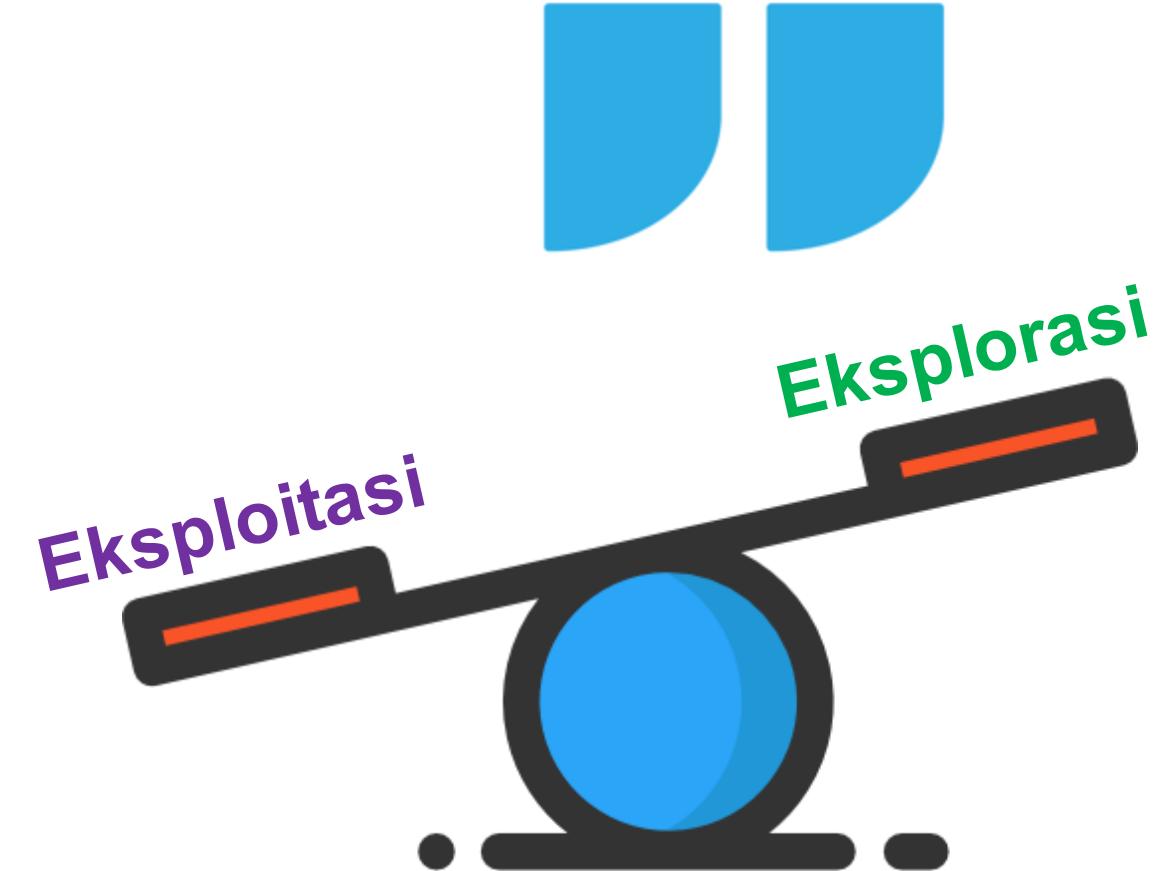
Ilustrasi Reward pada Pergerakan Agent RL (cont'd)

- Environment pada RL tidak mengajari agent untuk melakukan ini dan itu.
- Tetapi, agent melakukan aksi/pergerakan sendiri, lalu agent diberikan *reward* terhadap performa aksi/pergerakannya tersebut. **Disinilah proses belajar dalam RL terjadi.**
- Mekanisme *reward* tersebut, secara tidak langsung, akan mendorong agent untuk mencari *reward* positif sebanyak-banyaknya.
- Reward dapat diberikan setiap langkah, atau dapat juga diberikan setelah beberapa langkah.



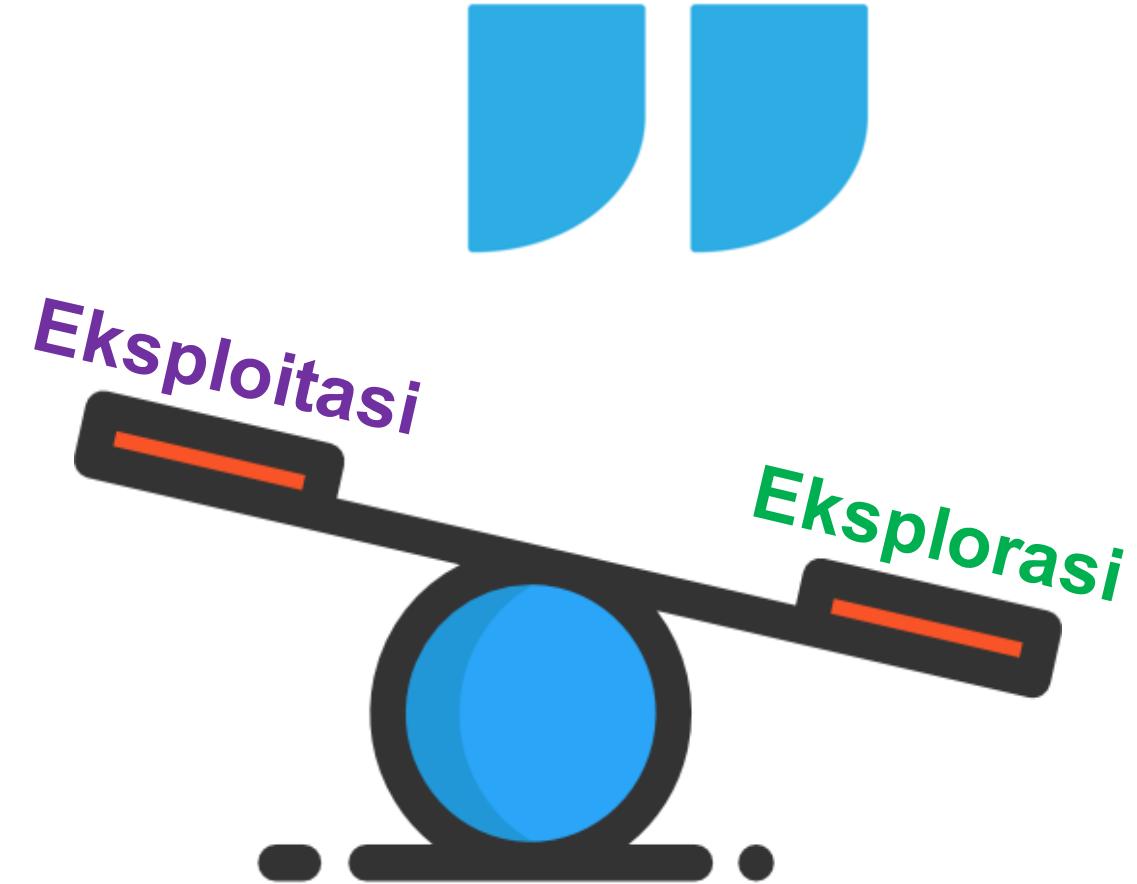
Implikasi Reward pada Algoritme RL

- Penentuan reward menentukan bagaimana algoritme RL melakukan eksplorasi dan eksplorasi.
- Eksplorasi adalah aksi algoritme RL dalam menggunakan aksi sebelumnya yang mendapat reward positif.
- Eksplorasi adalah aksi algoritme RL dalam melakukan aksi berbeda untuk mencari reward positif yang lain.



Implikasi Reward pada Algoritme RL (cont'd)

- Ada pitfall (jebakan) untuk ketidaktepatan dalam menentukan reward, menentukan hasil keluaran.
- Algoritme RL yang lebih dominan **eksplorasi**, ada kemungkinan peluang **solusi baik** lain akan terlewatkan
- Algoritme RL yang lebih dominan **eksplorasi**, ada kemungkinan agent RL tersebut akan mendapatkan banyak peluang **solusi buruk**.
- Solusi: Penerapan metode optimasi.



Contoh Algoritme RL



Permainan **Atari Breakout**



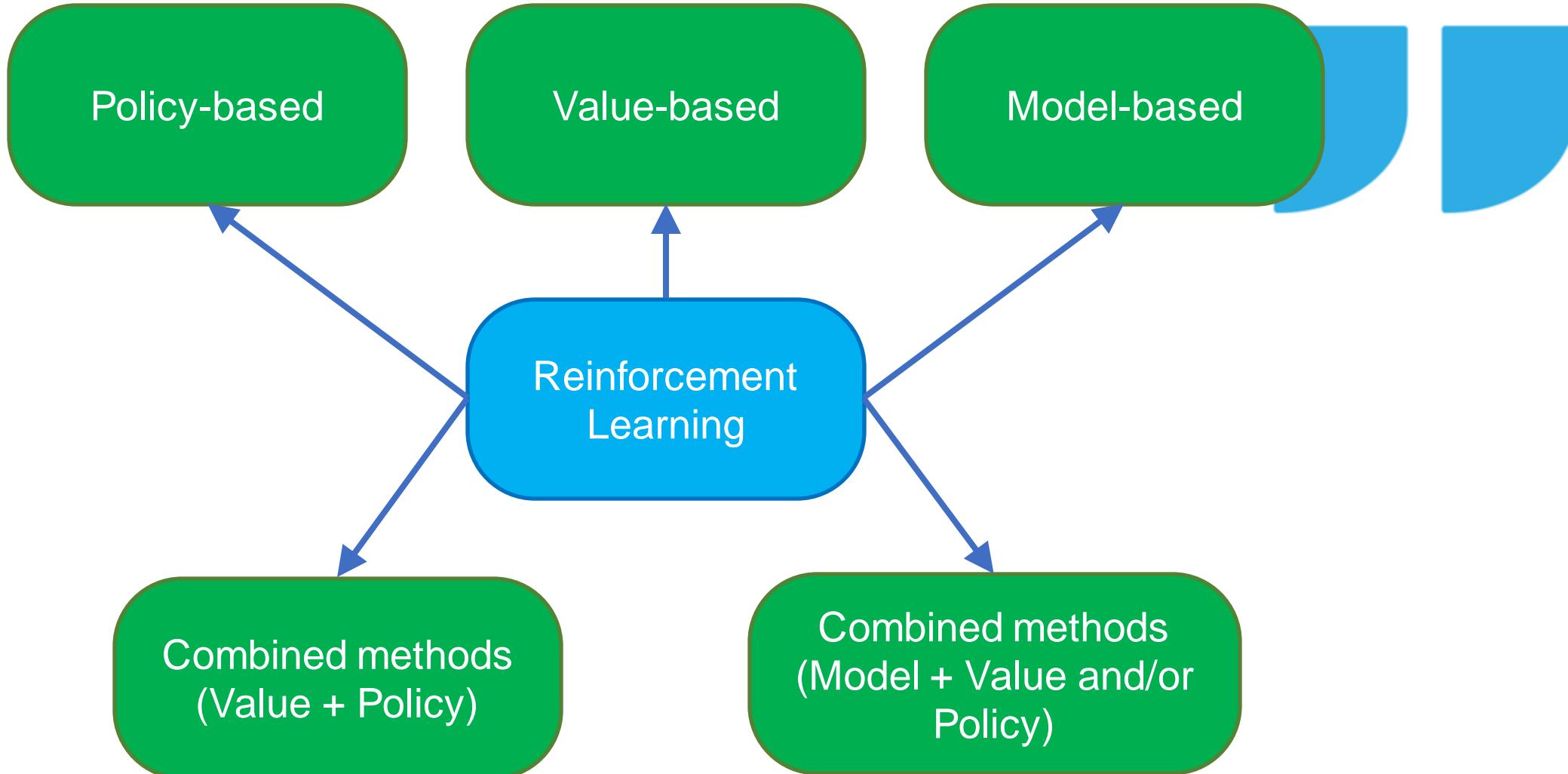
Seperti apa
environment algoritme
dari permainan **Atari
Breakout**?

Contoh Algoritme RL (cont'd)

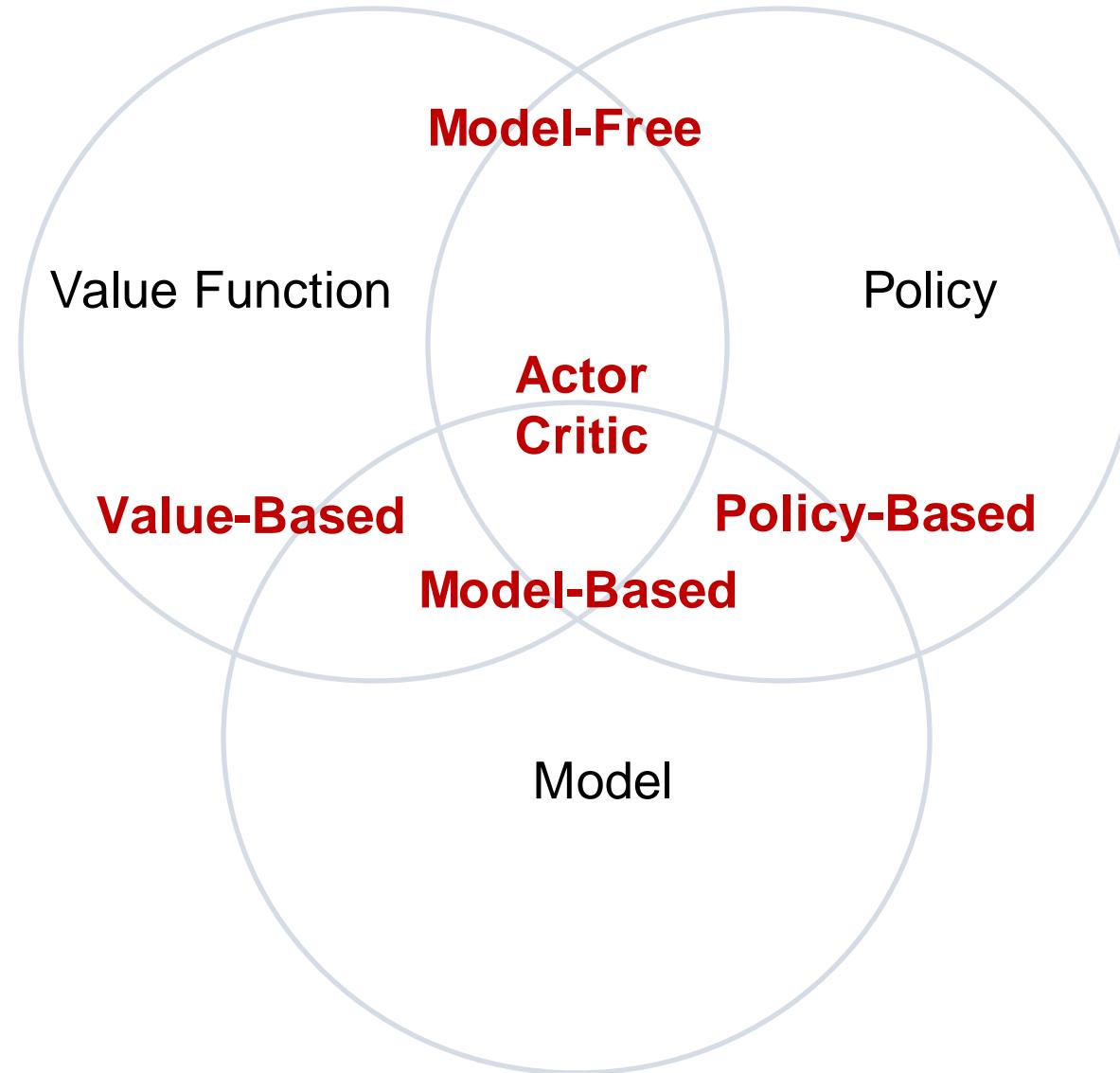
1. **Objective:** Memaksimalkan skor permainan
2. **State:** Gambar RGB dengan resolusi 160x210 px
3. **Action:** Integer dari himpunan {0, 1, 2, 3} yang memetakan pengendalian aksi dari game {no-action, launch the ball, move right, move left}
4. **Reward:** Skor permainan
5. **Termination:** Kalah permainan



Kategori Implementasi Algoritme RL



Taksonomi RL Agent





03

Elements & Environment RL

Elemen pada *Reinforcement Learning*

Di luar dari *Agent* dan *Environment* yang merupakan sebagai element utama, ada 4 sub elemen sebagai penyusun utama sistem *Reinforcement Learning*:

Elemen utama:

1. Agent
2. Environment

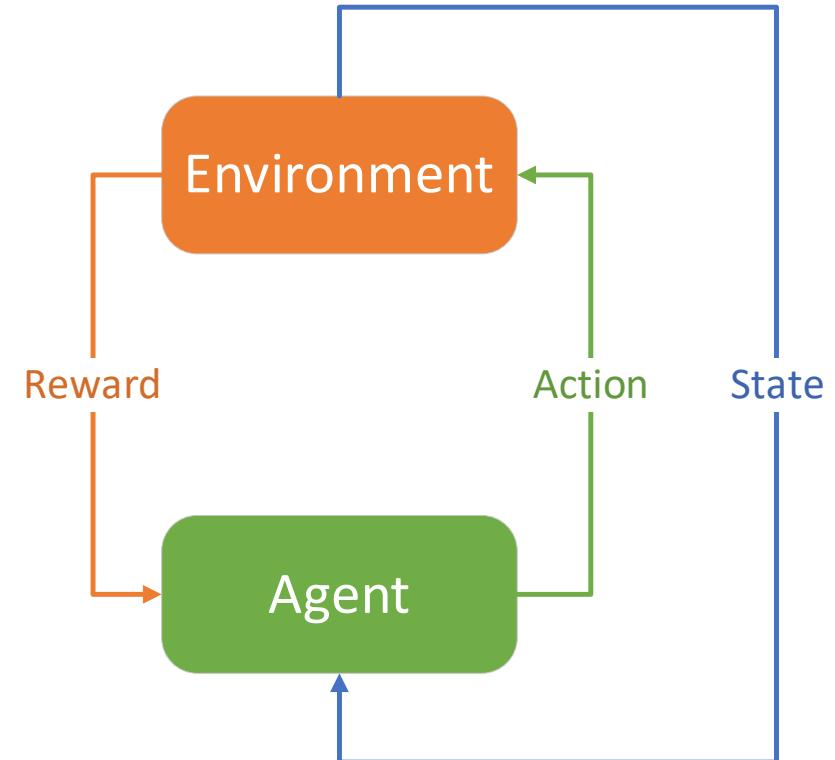
Sub Elemen utama:

1. Policy
2. Reward signal
3. Value function
4. Model environment (optional)

Elemen pada Reinforcement Learning

Agent dan Environment (next...)

- **Agent**, yang dimaksud dengan agent adalah perangkat atau software (perangkat lunak) yang dapat belajar dari *environment*.
- **Environment**, sedangkan yang dimaksud dengan *environment* adalah segala sesuatu yang ada di luar agent yang dimana sebagai tempat agent untuk melakukan *exploration* dan *exploitation*.



Sub Elemen pada *Reinforcement Learning*

Policy and Reward

- ***Policy***, *rules* (aturan) atau strategi yang digunakan oleh *agent* untuk melakukan *action* (**A**) selanjutnya, berdasarkan *state* (**S**) saat ini.
- ***Reward (R) Signal***, *feedback* atau umpan balik untuk *agent*. *Reward* dapat bernilai positif (berupa hadiah) atau negatif (berupa hukuman) dan juga nol (tidak ada tindakan apapun terhadap *agent*).

Sub Elemen pada *Reinforcement Learning*

Value function

- ***Value function (V)*** s , total nilai jangka panjang yang diharapkan (*expected long-term return without discount*) dari *state* saat ini di bawah *policy* π . *Value* ini adalah kebalikan dari *short-term reward (R)*.
 - *Optimal Value Function* adalah sebuah fungsi yang memiliki nilai tertinggi untuk semua *state* dibandingkan dengan fungsi nilai lainnya.
 - *Optimal Policy* adalah *policy* (kebijakan) yang memiliki fungsi nilai optimal.

Sub Elemen pada *Reinforcement Learning*

Model Environment

- **Model Environment**, segala sesuatu yang meniru perilaku environment. Atau, secara umumnya, sebuah kesimpulan tentang bagaimana perilaku dari *environment*.

Model Environment dibagi menjadi dua, yaitu:

- Model Based RL
- Model Free RL

Model Based RL vs Model Free RL

Dapatkanlah melihat perbedaan dari kedua game ini dari sudut pandang model environment-nya?

Game Go



VS

Dota 2



Model Based RL vs Model Free RL

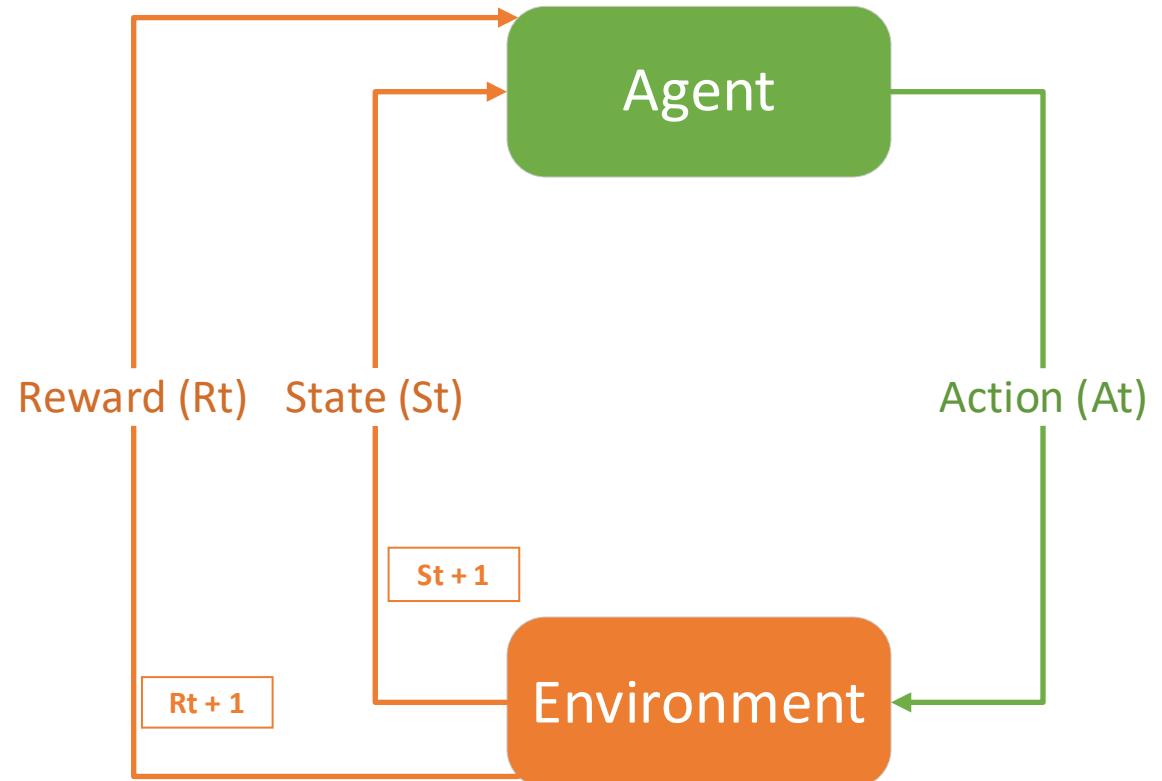
Dalam memecahkan persoalan yang berhubungan dengan *Reinforcement Learning*, ada dua metode yang dipakai, yaitu *model base* dan *model free*, keduanya didasarkan pada bagaimana lingkungannya yang dihadapi oleh *agent*.

- ❖ **Model Based RL**, model ini adalah model dasar atau paling sederhana dan memiliki perencanaan terhadap tindakannya, atau dapat juga dimaknai sebagai *agent* mengeksplorasi informasi yang dipelajari sebelumnya untuk menyelesaikan tugasnya.
- ❖ **Model Free RL**, sedangkan model ini adalah kebalikan dari model *Model Based RL*, karena *agent* belajar dari lingkungan melalui metode *trial and error* untuk memperoleh pengalamannya.

Environment RL

Agent environment interface

Agent adalah software agents yang melakukan aksi. Pada waktu t kemduain berpindah dari suatu state S_t ke state lainnya S_{t+1} . Sebagai imbalan agent akan mendapatkan reward berupa angka numerik atas setiap tindakannya dari environment.



Environment RL

Tipe-tipe Environment RL:

- Deterministic environment
- Stochastic environment
- Fully observable environment
- Partially observable environment
- Discrete environment
- Continuous environment
- Episodic and non-episodic environment
- Single and multi-agent environment

Tipe-tipe Environment RL

□ Deterministic environment vs Stochastic environment

Suatu *environment* dikatakan deterministik ketika hasilnya dapat diketahui berdasarkan *state* saat ini. Misalnya, pada permainan catur, kita dapat mengetahui hasilnya setelah pemain (pion, raja, banteng, kuda, dll) mana yang dipindahkan.

Sedangkan suatu *environment* dikatakan *Stochastic* ketika hasilnya tidak dapat diketahui berdasarkan *state* saat ini, dan environment tipe ini adalah kebalikan dari *deterministic*. Contohnya pada saat pelemparan dadu, ada kemungkinan sangat besar kita tidak dapat mengetahui angka berapa yang akan muncul.

□ Fully observable vs Partially observable environment

Fully observable environment adalah ketika *agent* dapat menentukan *state* dari sistem setiap saat, dengan *state* dari sistem dapat diamati seluruhnya. Misalnya, dalam permainan catur, posisi pemain kita dapat kapapun dan dimanapun kita pindahkan, sehingga kita dapat menentukan langkah paling optimal pada setiap kali memindahkan pemain.

Sedangkan *Partially observable*, ketika *agent* tidak dapat menentukan *state* sistem setiap saat. Misalnya, dalam permainan poker, kita tidak tahu kartu apa yang dimiliki oleh lawan.

Tipe-tipe Environment RL (Next...)

□ Discrete environment vs Continuous environment

Discrete environment, ketika *state* untuk berpindah dari *state* ke *state* lainnya terbatas. Misalnya, dalam permainan catur, dimana hanya memiliki satu set Gerakan terbatas.

Sedangkan *Continuous environment*, ketika *state* untuk berpindah dari *state* ke *state* lainnya tidak terbatas. Misalnya, dalam kasus menentukan rute jalan menuju suatu tempat dengan jalan yang berbeda-beda.

Tipe-tipe Environment RL (Next...)

Episodic and non-episodic environment

Environment episodik disebut juga sebagai lingkungan non-sekuensial. Pada *environment* episodik, tindakan *agent* saat ini tidak akan mempengaruhi tindakan di masa depan, sedangkan dalam *environment* non-episodik tindakan *agent* saat ini mempengaruhi tindakan di masa depan dan juga disebut **sequential environment**. Artinya, *agent* melakukan tugas-tugas independen di lingkungan episodik, sedangkan dalam lingkungan non-episodik semua tindakan *agent* terkait.

Single and multi-agent environment

Single environment, hanya memiliki *agent* tunggal, sedangkan *multi-agent environment* memiliki lebih dari 1 *agent* atau bahkan bisa lebih. *Multi-agent environment* sering digunakan pada saat melakukan tugas-tugas kompleks. *Agent* dengan *environment* yang berbeda dapat saling berkomunikasi, dan *multi-agent environment* sebagian besar dapat berubah menjadi stokastik karena memiliki tingkat ketidakpastian yang lebih besar.

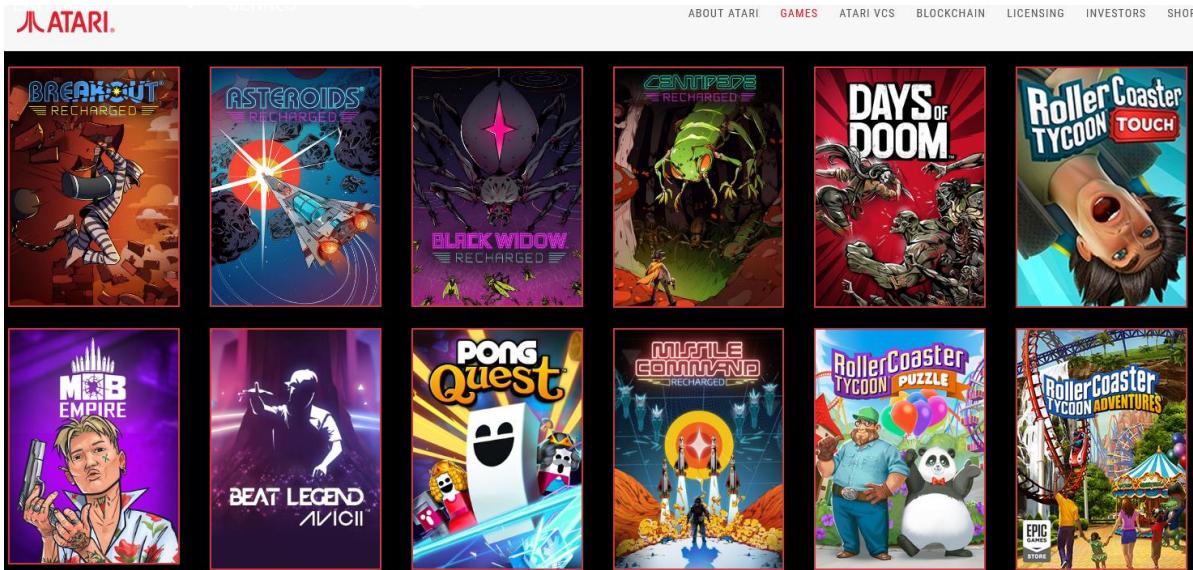


04 Aplikasi RL

- AI of Game
- Robotics
- Optimization

Reinforcement Learning for AI in Games

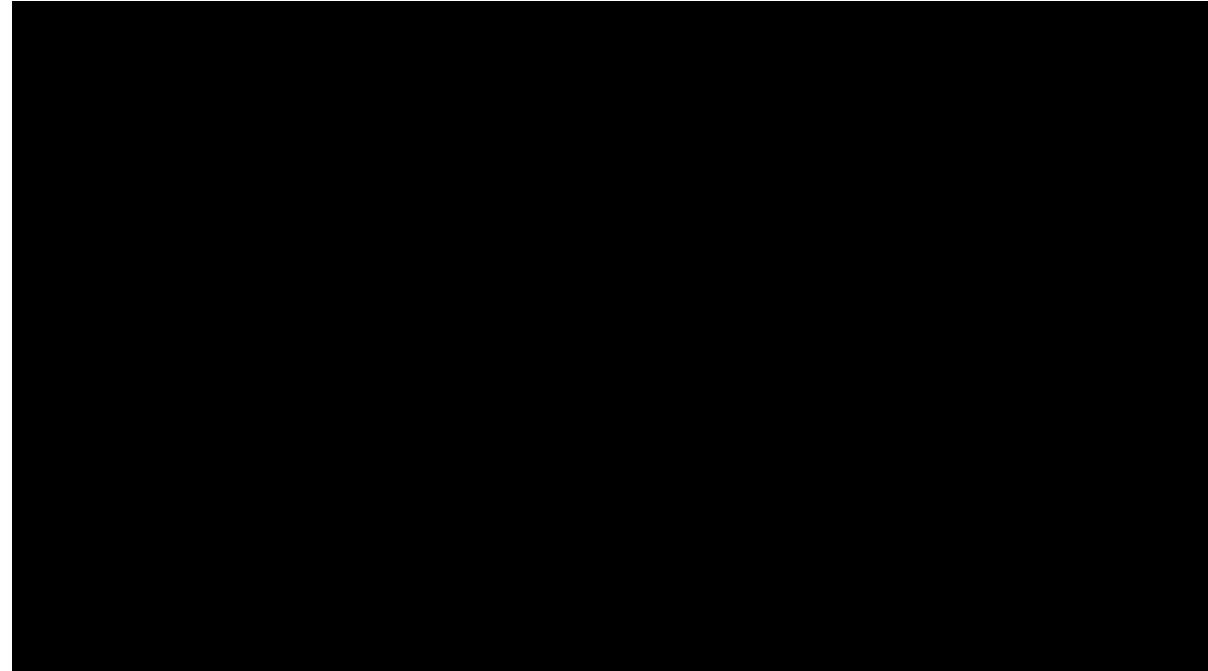
- Atari Games



Source:

<https://www.atari.com/atari-games/>

- OpenAI Five : Dota



Source:

<https://openai.com/blog/openai-five/>

Reinforcement Learning for AI of Game

- Examples of Building AI for Snake Game using Reinforcement Learning

Mendefinisikan state yang mungkin terjadi
(11 states)

```
state = [
    # Danger straight
    (dir_r and game.is_collision(point_r)) or
    (dir_l and game.is_collision(point_l)) or
    (dir_u and game.is_collision(point_u)) or
    (dir_d and game.is_collision(point_d)),

    # Danger right
    (dir_u and game.is_collision(point_r)) or
    (dir_d and game.is_collision(point_l)) or
    (dir_l and game.is_collision(point_u)) or
    (dir_r and game.is_collision(point_d)),

    # Danger left
    (dir_d and game.is_collision(point_r)) or
    (dir_u and game.is_collision(point_l)) or
    (dir_r and game.is_collision(point_u)) or
    (dir_l and game.is_collision(point_d)),

    # Move direction
    dir_l,
    dir_r,
    dir_u,
    dir_d,

    # Food location
    game.food.x < game.head.x, # food left
    game.food.x > game.head.x, # food right
    game.food.y < game.head.y, # food up
    game.food.y > game.head.y # food down
]
print(state[0],state[1],state[2],state[3],state[4],state[5],state[6],state[7],state[8],state[9],state[10])
```

Mendefinisikan action yang ada
(3 actions : Straight, Turn right, Turn left)

```
def _move(self, action):

    """
    The Snake Can only move [Straight, Right, Left]
    """

    clock_wise = [Direction.RIGHT, Direction.DOWN, Direction.LEFT, Direction.UP]
    idx = clock_wise.index(self.direction)

    if np.array_equal(action, [1, 0, 0]):
        new_dir = clock_wise[idx] # no change
    elif np.array_equal(action, [0, 1, 0]):
        next_idx = (idx + 1) % 4
        new_dir = clock_wise[next_idx] # right turn r -> d -> l -> u
    else: # [0, 0, 1]
        next_idx = (idx - 1) % 4
        new_dir = clock_wise[next_idx] # left turn r -> u -> l -> d

    self.direction = new_dir

    x = self.head.x
    y = self.head.y
    if self.direction == Direction.RIGHT:
        x += BLOCK_SIZE
    elif self.direction == Direction.LEFT:
        x -= BLOCK_SIZE
    elif self.direction == Direction.DOWN:
        y += BLOCK_SIZE
    elif self.direction == Direction.UP:
        y -= BLOCK_SIZE

    self.head = Point(x, y)
```

Reinforcement Learning for AI of Game

```
def get_action(self, state):
    # random moves: tradeoff exploration / exploitation
    self.epsilon = 80 - self.n_games
    final_move = [0,0,0]
    if random.randint(0, 200) < self.epsilon:
        move = random.randint(0, 2)
        final_move[move] = 1
    else:
        state0 = torch.tensor(state, dtype=torch.float)
        prediction = self.model(state0)
        move = torch.argmax(prediction).item()
        final_move[move] = 1

    return final_move
```

- Mendefinisikan nilai gradient policy untuk menentukan kapan akan melakukan explorasi maupun exploitasi.

```
target = pred.clone()
for idx in range(len(done)):
    Q_new = reward[idx]
    if not done[idx]:
        #BELLMAN Equation
        Q_new = reward[idx] + self.gamma * torch.max(self.model(next_state[idx]))

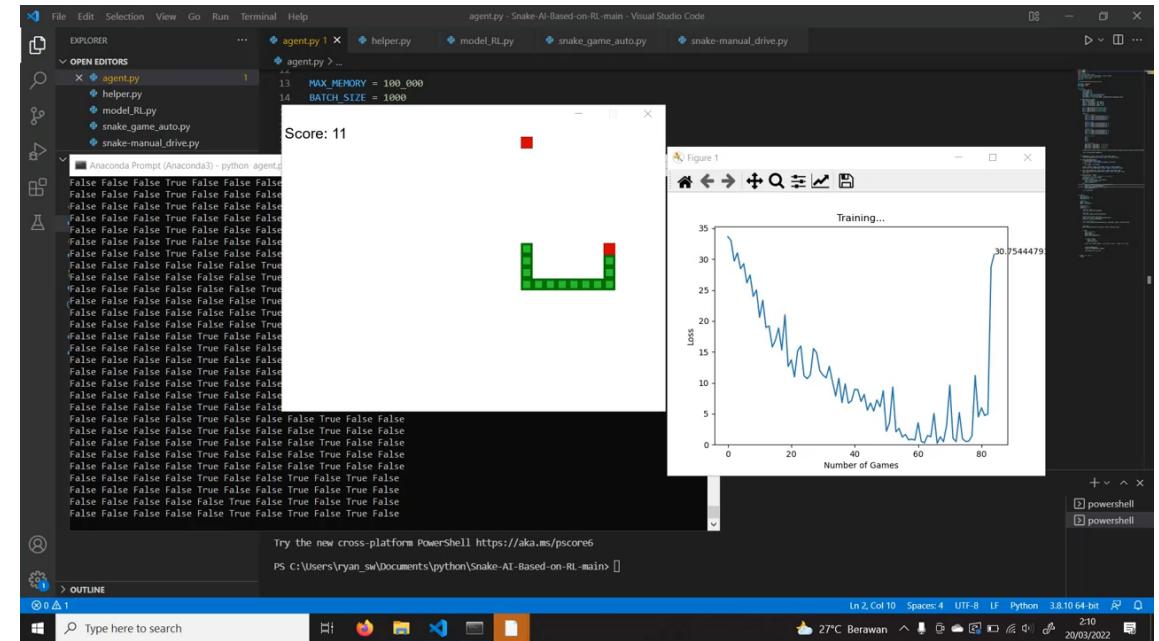
    target[idx][torch.argmax(action[idx]).item()]= Q_new
```

- Mendefinisikan model Reward yang akan digunakan.

Reinforcement Learning for AI in Games

- Kinerja saat pelatihan episode awal
- Setelah pelatihan berjalan 80 episode

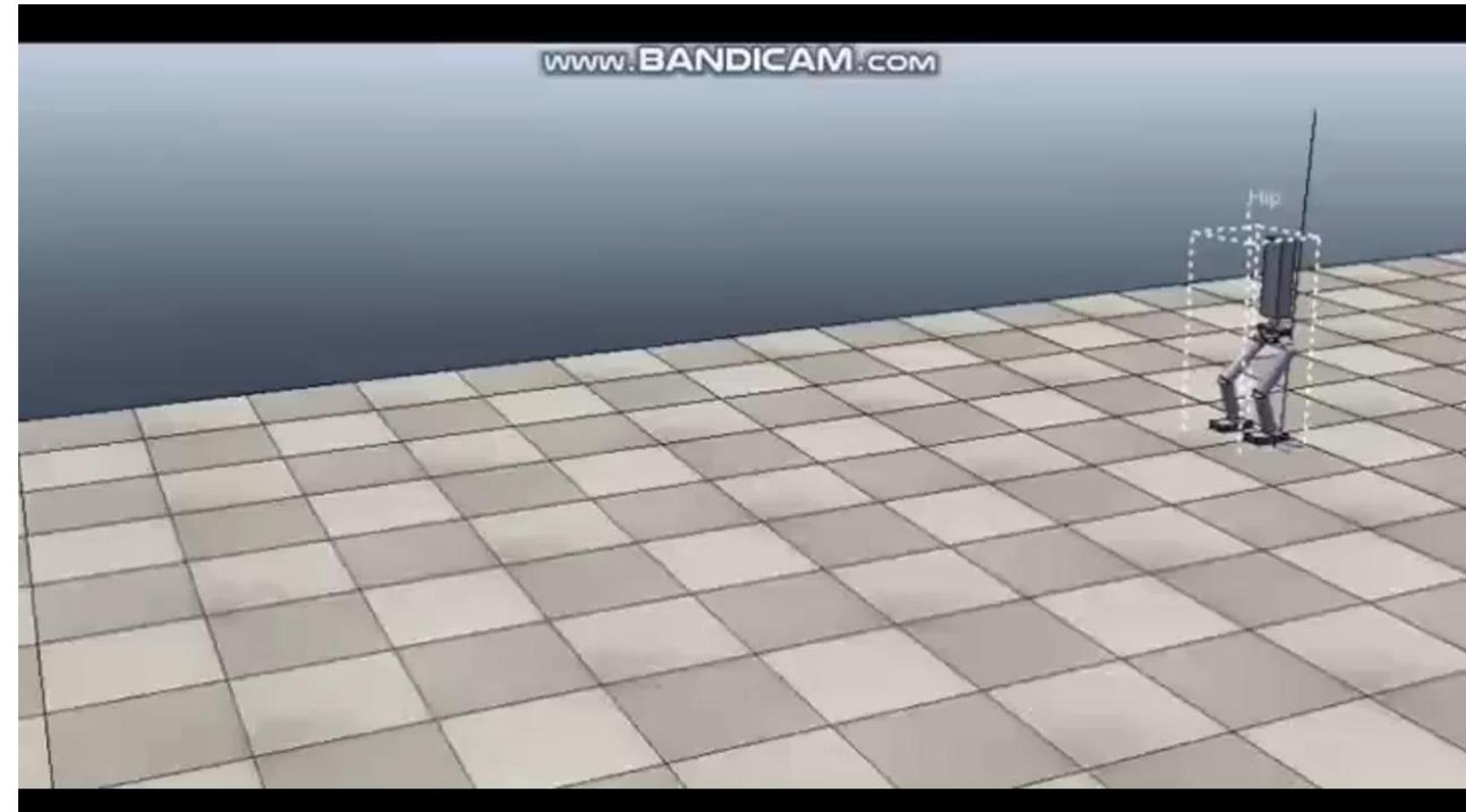
A screenshot of Visual Studio Code showing the initial state of a reinforcement learning project for a Snake game. The code editor displays several files: agent.py, helper.py, model_RL.py, snake_game_auto.py, and snake-manual_drive.py. The terminal window shows the command 'python agent.py' being run, and the output indicates the AI is still learning, with many 'False' and 'True' values printed. The status bar at the bottom shows the Python version as 3.8.10 64-bit.



Reinforcement Learning for Robotics

- Examples of building trajectory generation for Humanoid Robot using Reinforcement Learning.

- State : Jumlah joint atau derajat kebebasan (DoF) dari suatu robot dalam bentuk sudut radian.
- Action: Posisi koordinat kartesian X,Y,Z pada setiap kaki (6 actions).



Reinforcement Learning for Robotics

- Examples of Self Driving Car using Reinforcement Learning.

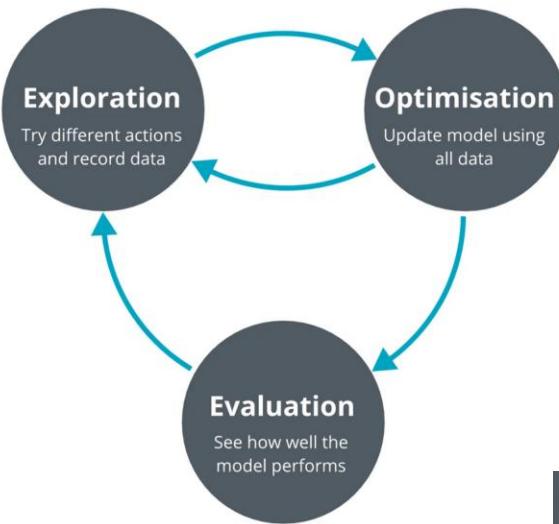
- Motion planning, trajectory optimization, dynamic pathing, controller optimization, dan scenario-based learning policies merupakan salah satu bagian dari aplikasi dari self-driving car dengan menggunakan reinforcement learning.
- Places, Driveable zones, Obstacle avoidance, Velocity, dan Direction merupakan beberapa variable yang bisa didefinisikan sebagai state dan action pada reinforcement learning model untuk self-driving car.

Source:

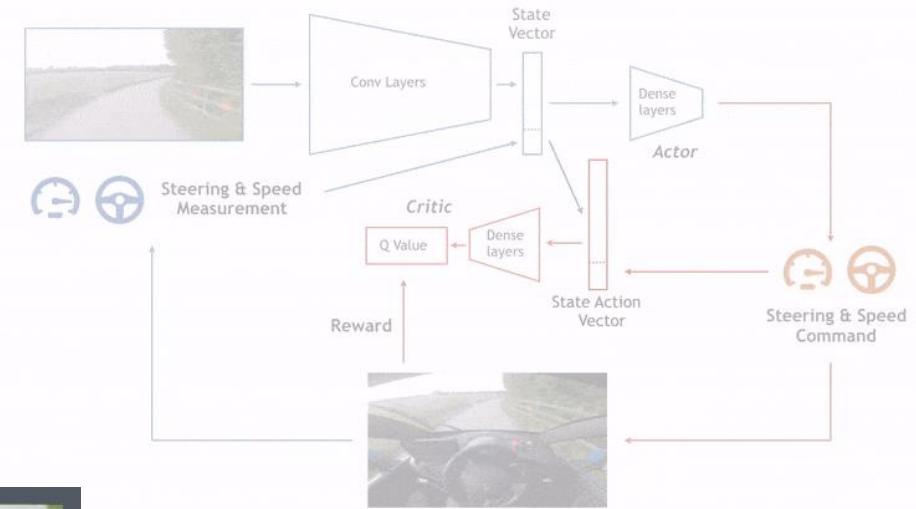
<https://wayve.ai/blog/learning-to-drive-in-a-day-with-reinforcement-learning/>

Reinforcement Learning for Robotics

deep deterministic policy gradients, DDPG



Self Driving Algorithm



Simulation Result



Source:

<https://wayve.ai/blog/learning-to-drive-in-a-day-with-reinforcement-learning/>

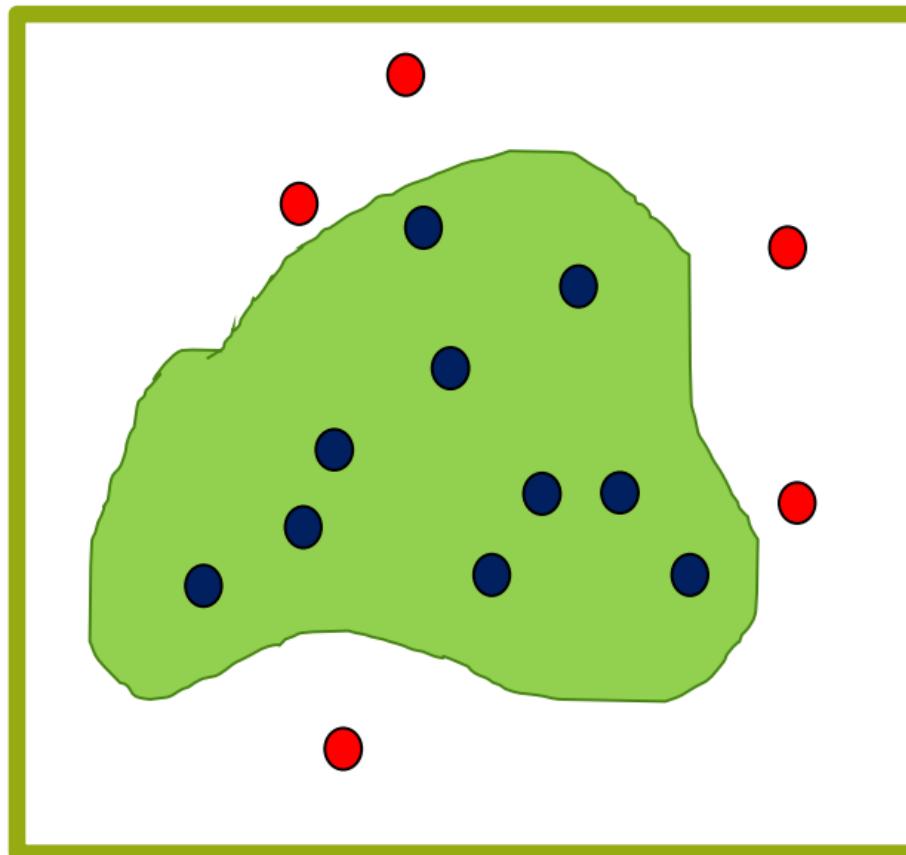
Reinforcement Learning for Optimization

Berikut beberapa contoh dari penerapan Reinforcement Learning untuk Optimasi:

1. Menentukan luasan suatu area dengan pendekatan optimasi monte carlo prediction.
2. Menentukan model suatu fungsi matematika dengan pendekatan optimasi monte carlo prediction
3. Membuat solusi dari permasalahan kasus Traveling Salesman Problem.
4. Membuat suatu sistem Simultaneous Locallization and Mapping (SLAM) untuk mengetahui posisi object terhadap lingkungan dalam koordinat lokal maupun koordinat global.

Reinforcement Learning for Optimization

- Determine the value of an area using monte carlo prediction



Reinforcement Learning for Optimization

- Monte Carlo Prediction for solving Traveling Salesman Problem

The screenshot shows the MATLAB IDE interface. The left pane displays the code editor for 'tsp3.m' located at F:\S2\Semester4\Permodelan dan Simulasi\week5\tsp3.m. The right pane shows the workspace variables and the command window output.

Code Editor (tsp3.m):

```
1 %Membaca data kota
2 load kota
3 nkota=size(kota,1);
4 maxIter=200;
5 pos=kota;
6 % Menggambar posisi kota
7 figure(1)
8 plot(pos(:,1),pos(:,2), 'o'), grid
9 hold on
10
11 %State awal
12 s=randperm(nkota);
13 s(nkota+1)=s(1);
14 tjarak=0;
15 for i=1:nkota
16     tjarak=tjarak+jarak(pos(s(i),:),pos(s(i+1),:));
17 end
18 jarak_min=tjarak;
19 s_min=s;
20
```

Command Window:

```
2
n =
```

Workspace:

Name
d
i
iter
jcover
jcover_m
k
maxIter
p
pmax
s
x
y

This presentation is made by



Wayan Dadang

Wijaya Yudha Atmaja

Dino Febriyanto S.Kom.

Ryan Satria Wijaya S.T.

Oetomo



References

- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2020
- Sudharsan Ravichandiran. Hands-On Reinforcement Learning with Python. 2018



Student Activities

1. Cari dan tuliskan satu contoh aplikasi/implementasi RL dibidang anda!
2. Jelaskan secara singkat mengapa aplikasi tersebut merupakan aplikasi berbasis RL, jelaskan berdasarkan kesesuaian dengan karakteristik RL!
3. Dari aplikasi tersebut, tentukan: (a) Objective (b) State (c) Action (d) Reward (e) Termination.
4. Berdasarkan 3 nomor diatas, utarakan/jelaskan pemahaman anda tentang RL, terkait karakteristik, derajat kemudahan/kesulitan untuk diaplikasikan menyelesaikan masalah riil dst.





TERIMA KASIH

Orbit Future Academy

PT Orbit Ventura Indonesia
Center of Excellence (Jakarta Selatan)
Gedung Veteran RI, Lt.15
Unit Z15-002, Plaza Semanggi
Jl. Jenderal Sudirman Kav.50, Jakarta
12930, Indonesia

- Jakarta Selatan/Pusat
- Jakarta Barat/BSD
- Kota Bandung
- Kab. Bandung
- Jawa Barat

Hubungi Kami

Director of Sales & Partnership
ira@orbitventura.com
+62 858-9187-7388

Social Media

-  Orbit Future Academy
-  @OrbitFutureAcademyIn1
-  OrbitFutureAcademy
-  Orbit Future Academy

AI Mastery Course



Modul Domain AI

Reinforcement Learning
Belajar dengan Penguatan

Bagian

Markov Decision Process (MDP)
dan Dynamic Programming



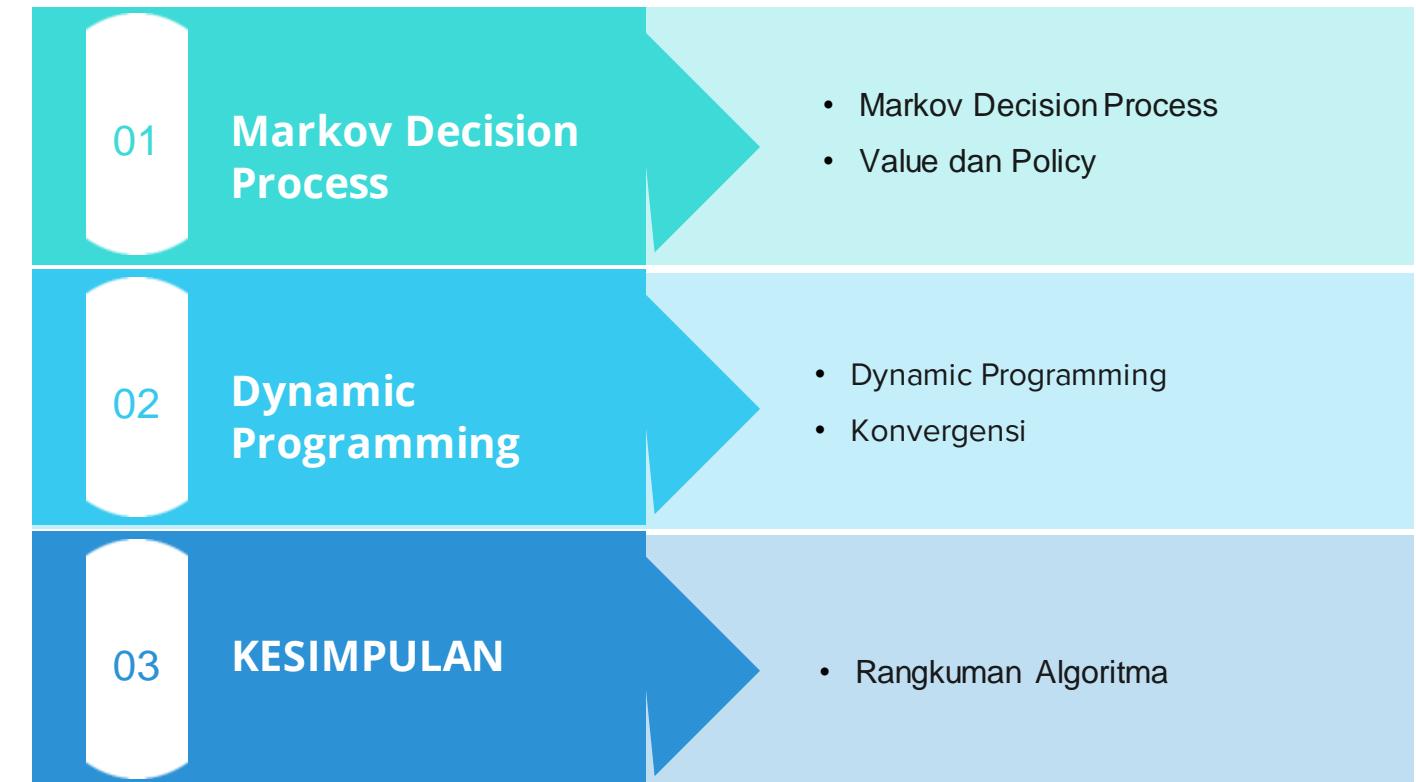


Learning Objectives

1. Mengetahui Markov Decision Proses,
2. Bisa menyusun MDP dalam merumuskan RL secara formal
3. Menentukan Value sebuah Policy.
4. Mencoba membuat Dynamic Programming tipe Value Iteration



Agenda



Markov Property

“The future is independent of the past given the present”

Sebuah keadaan S_t bisa disebut *Markov* jika dan hanya jika:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, S_2, \dots, S_t)$$

- Sebuah *state* memiliki informasi dari sejarah
- Ketika *state* diketahui, sejarah dapat diabaikan

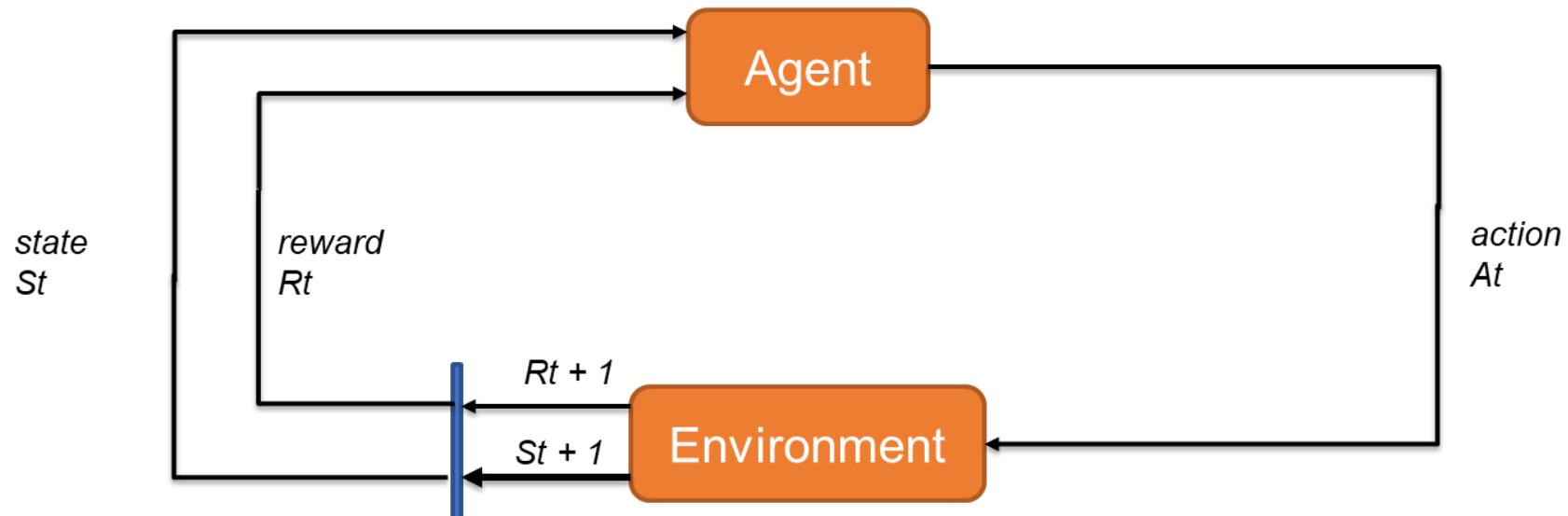
Apa itu MDP?

Markov Decision Process merupakan sebuah *tuple* (S, A, P, R, γ) .

Dimana:

- S merupakan *state*
- A merupakan *action*
- P merupakan *state transition probability function (transition probability)*
- R merupakan *reward function*
- γ merupakan *discount factor* ($\gamma \in [0, 1]$)

Nah, apabila sudah mengenal MDP, kita sudah dapat membahas RL



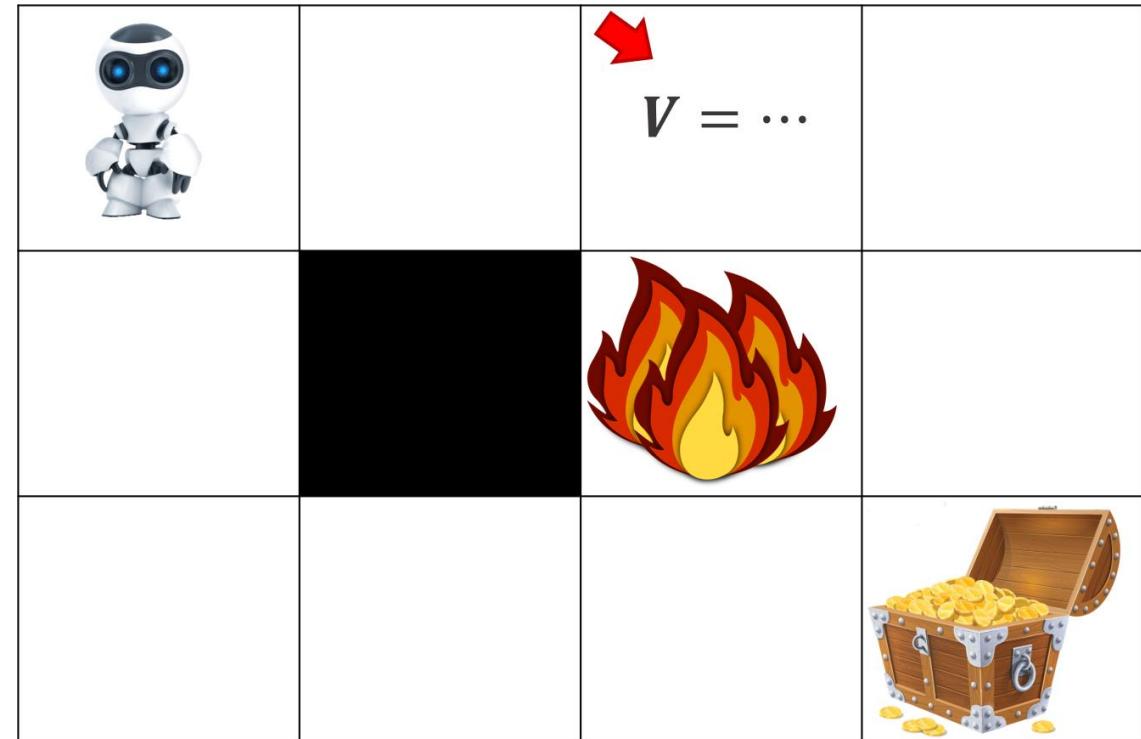
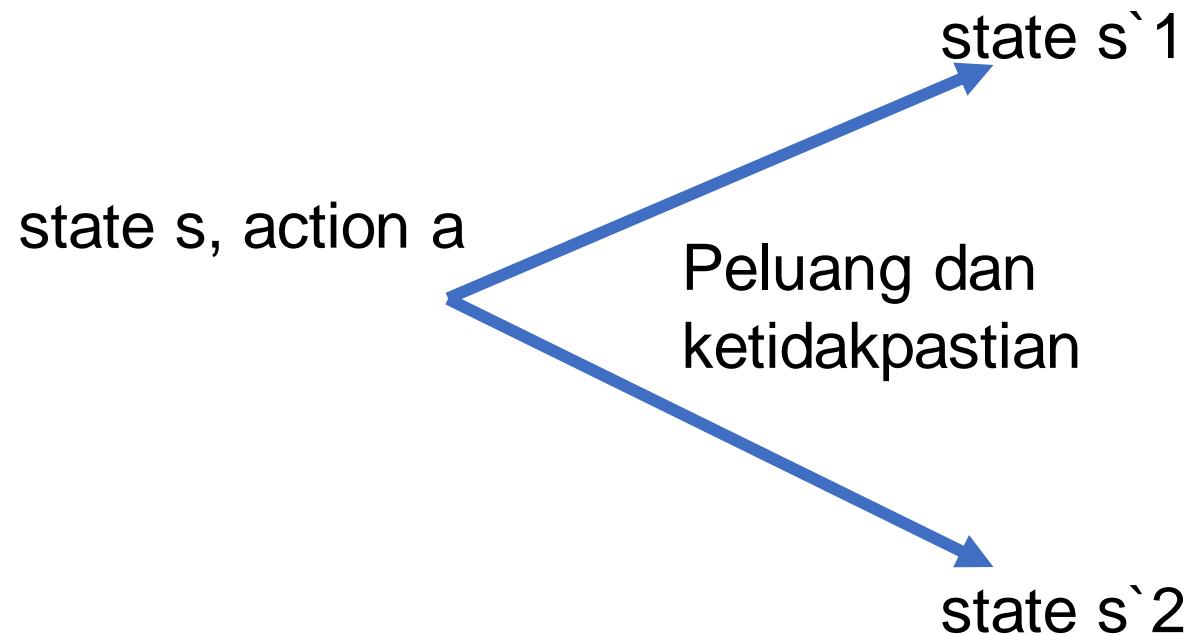
Apa itu MDP? (Cont'd...)

- *Markov Decision Processes* mendeskripsikan secara formal lingkungan untuk RL
- Secara spesifik biasanya dibuat saat environment *fully observable*
- Hampir semua RL problems dapat diformalisasi menggunakan MDP
 - Optimal Control
 - Partially Observable problems
 - Bandits problem

Bayangkan Anda Pergi Kuliah

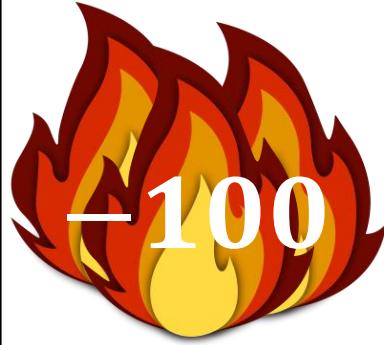
- Moda **transportasi** apa yang akan anda gunakan?
 - Jalan kaki?
 - Sepeda?
 - Angkutan Umum?
 - Taksi online?
 - Taksi

Ketidakpastian dalam Dunia Nyata



Aplikasi RL dalam Dunia Nyata

- **Robotik:** Memutuskan bergerak kemana.
- **Alokasi Sumber Daya:** Memutuskan apa yang diproduksi
- **Pertanian :** Menentukan apa yang ditanam, tapi tidak tahu cuaca dan hasil tanam

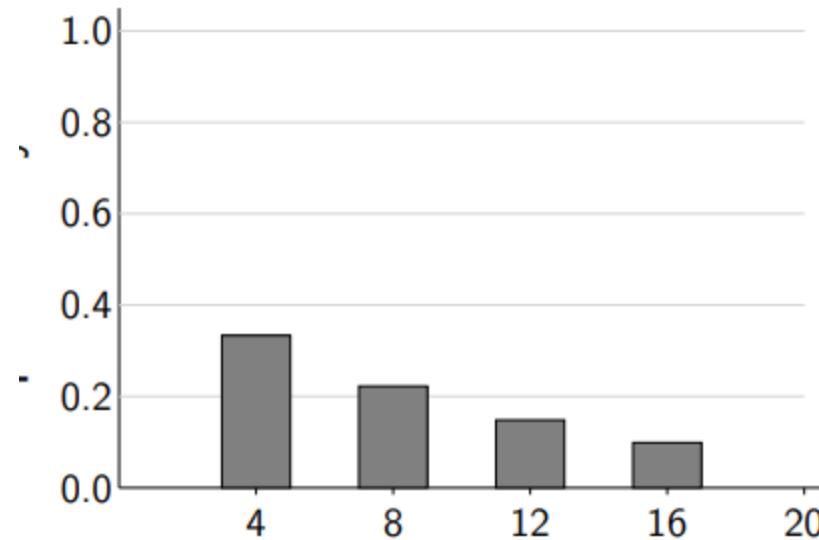
		 $V = \dots$	
		 -100	
			 +100

Contoh: Permainan Dadu

- Untuk setiap ronde: $r = 1, 2, \dots$
 - Bisa memilih untuk **main** atau **keluar**
 - Jika **keluar**, mendapatkan 10 koin dan permainan berakhir
 - Jika **main**, mendapatkan 4 koin dan dadu dilempar
 - Jika dadu keluar angka 1 atau 2, permainan berakhir
 - Selain itu, lanjutkan ronde berikut

Rewards

- Jika mengikuti *policy* “main”

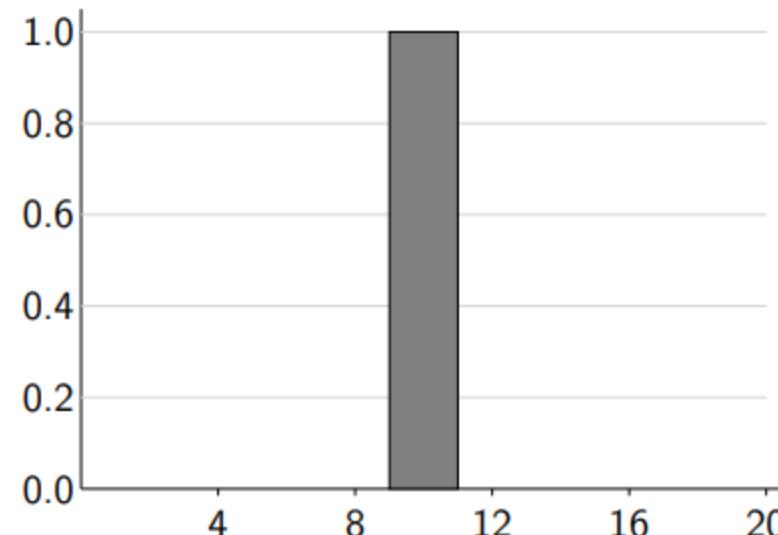


- Expected return:

$$\frac{1}{3}(4) + \frac{2}{3} \cdot \frac{1}{3}(8) + \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}(12) + \dots = 12$$

Rewards

- Jika mengikuti *policy* “keluar”



- Expected return:

$$1(10) = 10$$

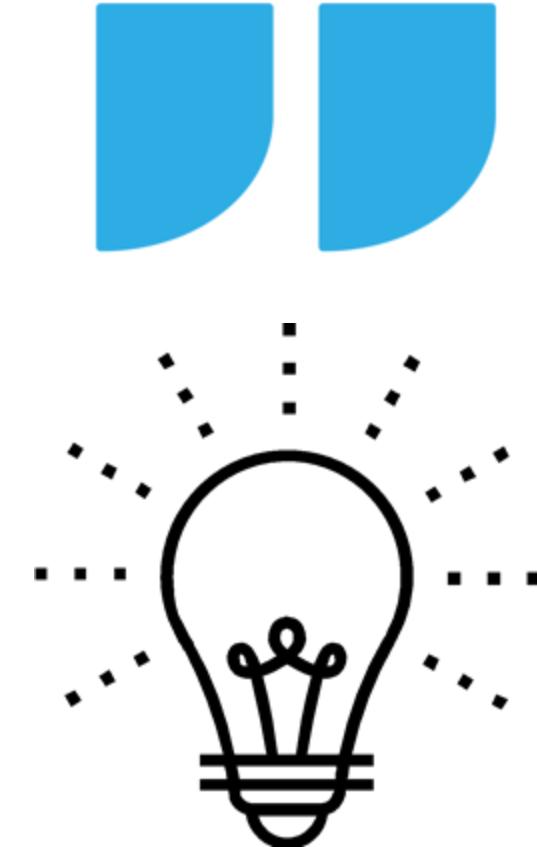
MDP untuk Permainan Dadu

- Untuk setiap ronde: $r = 1, 2, \dots$
 - Bisa memilih untuk **main** atau **keluar**
 - Jika **keluar**, mendapatkan 10 koin dan permainan berakhir
 - Jika **main**, mendapatkan 4 koin dan dadu dilempar
 - Jika dadu keluar angka 1 atau 2, permainan berakhir
 - Selain itu, lanjutkan ronde berikut

Question ?

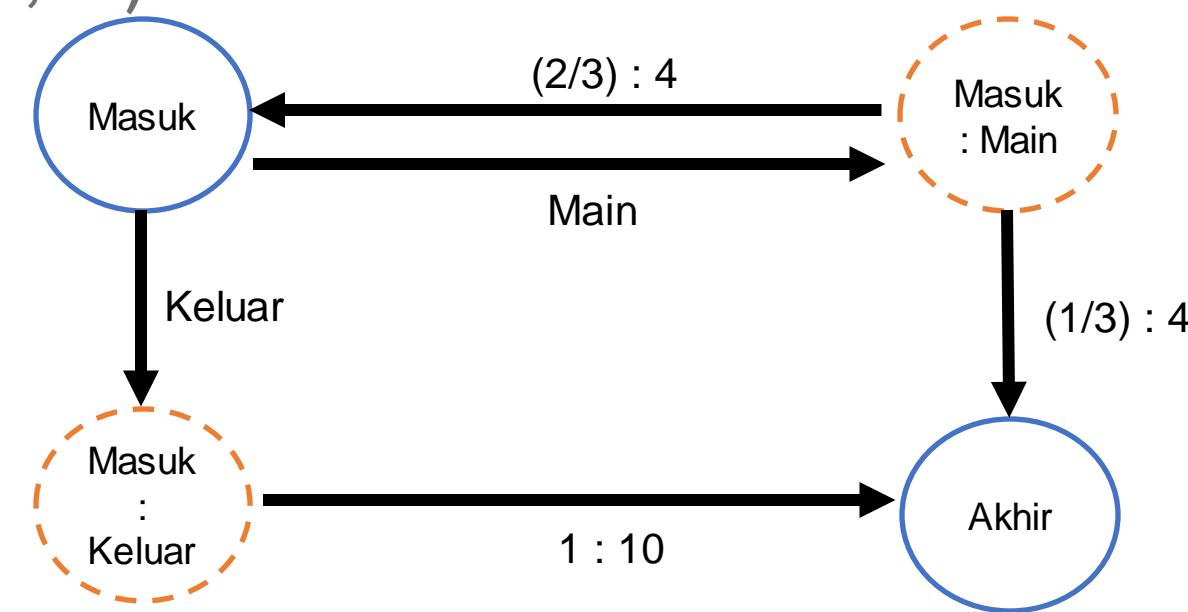
The only stupid question is the one
you were afraid to ask but never did.

-Rich Sutton



MDP untuk Permainan Dadu

- State (s) node
- Action (a)
- State : Action q(s, a) node
- Probability P(s, a, s')
- Reward r(s, a, s')



Markov Decision Process

- States: himpunan dari states
- $s_{start} \in States$: state awal
- Actions(s): actions yang tersedia dari state s
- $P(s, a, s')$: probabilitas ke s' jika mengambil action a dari state s
- Reward (s, a, s') : idem dengan diatas
- $\text{isEnd}(s)$: apakah state terakhir
- $0 \leq \gamma \leq 1$ faktor diskon

Probabilitas (atau Transition)

- Transition Probabilitas $P(s, a, s')$ menentukan kemungkinan dari state yang didatangi dalam s' jika melakukan action a dari state s
- Contoh:

s	a	s'	P(s, a, s')
Masuk	Keluar	Akhir	1
Masuk	Main	Masuk	2/3
Masuk	Main	Akhir	1/3

Probabilitas memiliki jumlah 1

- Untuk setiap state s dan action a:

$$\sum_{s' \in States} P(s, a, s') = 1$$

s	a	s'	P(s, a, s')
Masuk	Keluar	Akhir	1
Masuk	Main	Masuk	2/3
Masuk	Main	Akhir	1/3

Matrik Probabilitas Transisi

s	a	s'	P(s, a, s')
Masuk	Keluar	Akhir	1
Masuk	Main	Masuk	2/3
Masuk	Main	Akhir	1/3

action = Main	Masuk	Akhir	action = Keluar	Masuk	Akhir
Masuk	2/3	1/3	Masuk	0	0
Akhir	0	1	Akhir	0	1

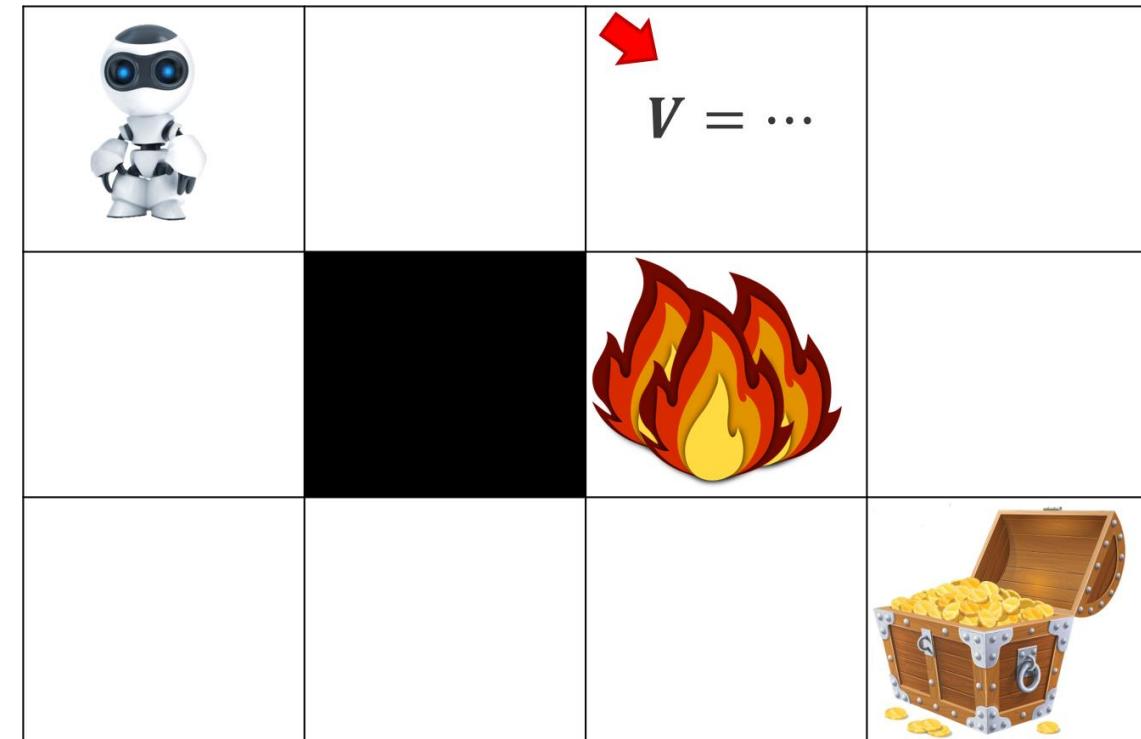
Contoh Transportasi

- Sebuah kota yang mempunyai nomor blok 1 sampai n
- Berjalan dari s ke $s+1$ membutuhkan waktu 1 menit
- Naik tram ajaib dari s ke $2s$ membutuhkan waktu 2 menit
- Bagaimana berpindah dari 1 ke n dalam waktu paling sedikit?
- **Tram punya kemungkinan gagal 0.5**
- Jika gagal, waktu habis 2 menit tapi tidak pindah blok

Solusi untuk MDP?

- *Policy* adalah fungsi π yang memetakan untuk setiap state $s \in States$ ke action $a \in Actions(s)$
- Contoh Robot pencari harta karun:

s	$\pi(s)$
(1,1)	Bawah
(2,1)	Bawah
(3,1)	Kanan
....



Evaluasi sebuah *policy*

- Mengikuti *policy* akan bisa menghasilkan **jalur yang acak** (mengapa?)
- ***Return* (*utility*)** dari sebuah *policy* adalah Jumlah dari *reward* selama mengikuti jalur (nilai yang acak)
- Contoh *policy* = “main” pada permainan dadu:

Jalur	Return
[masuk;main,4,akhir]	4
[masuk;main,4, masuk;main,4, masuk;main,4,akhir]	12
[masuk;main,4, masuk;main,4, akhir]	8
[masuk;main,4, masuk;main,4, masuk;main,4, masuk;main,4, akhir]	16
....

Return (Utility)

- Return adalah G_t total reward yang di-diskon dari setiap waktu t

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Jalur	Return
[masuk;main,4,akhir]	4
[masuk;main,4, masuk;main,4, masuk;main,4,akhir]	12
[masuk;main,4, masuk;main,4, akhir]	8
[masuk;main,4, masuk;main,4, masuk;main,4, masuk;main,4, akhir]	16
....

Diskon γ

- Definisi: return
- Jalur: $s_0, a_1, r_1, s_1, a_2, r_2, s_2, \dots$ (action, reward, state baru)
- Return dengan diskon adalah:

$$u_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots$$

- Diskon $\gamma = 1$ (sangat memperhatikan masa depan)
[masuk, masuk, masuk, masuk]: $4+4+4+4 = 16$
- Diskon $\gamma = 0$ (Hidup Cuma sekali)
[masuk, masuk, masuk, masuk]: $4+ 0.(4+0.(4+\dots))= 4$
- Diskon $\gamma = 0.5$ (Hidup yang seimbang)
[masuk, masuk, masuk, masuk]: $4+ 0.5(4+0.5(4+\dots))= 7.5$

Apa guna diskon? γ

Biasanya Markov reward dan MDP mengandung diskon, mengapa?

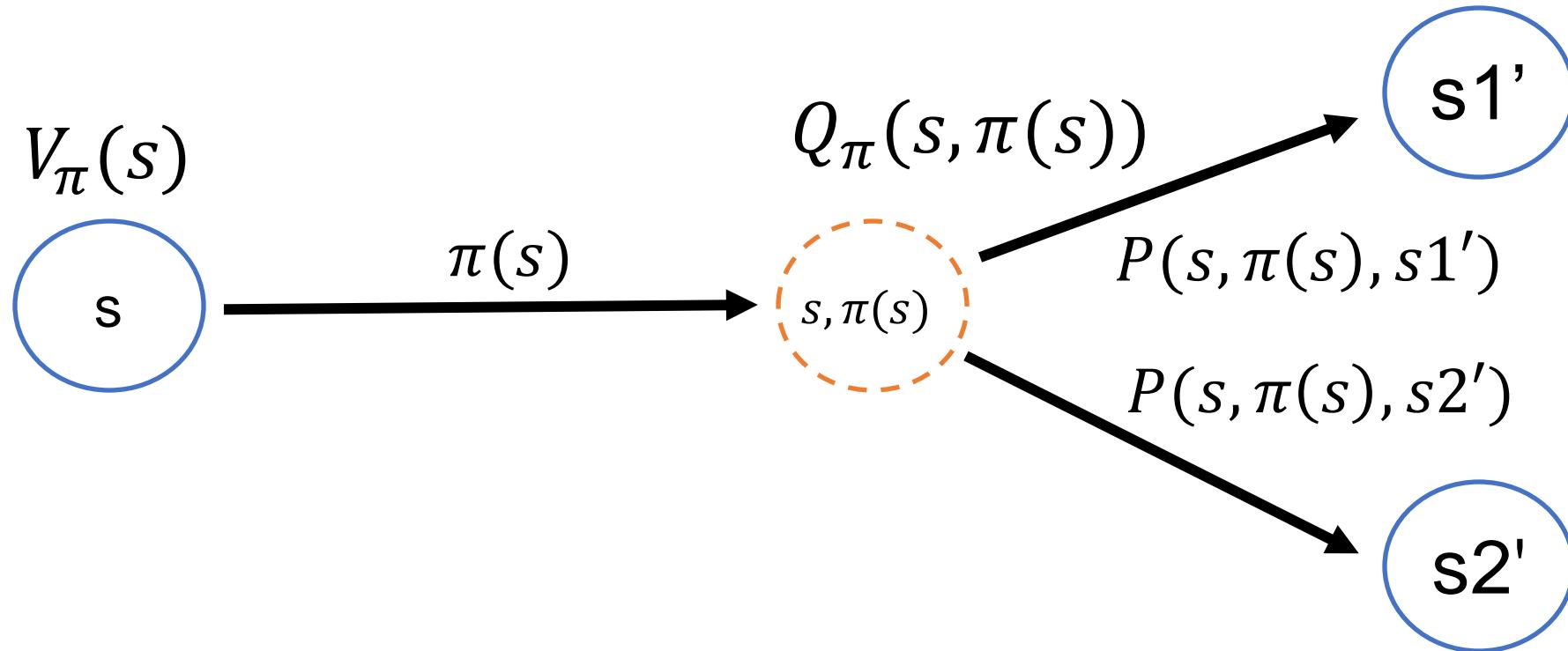
- Secara matematik memberikan konstanta untuk diskon itu mudah
- Menghindari return tak terhingga dalam proses Markov yang berbentuk siklus
- Masa depan yang tidak pasti yang mungkin tidak diwakili model
- Dalam hal finansial, reward langsung lebih menarik disbanding reward nanti
- Perilaku binatang / manusia biasanya lebih memilih reward langsung
- Tetap memungkinkan untuk menggunakan Markov reward yang tidak didiskon ($\gamma = 1$) (jika semua alur tidak berulang)

Recap and Question?

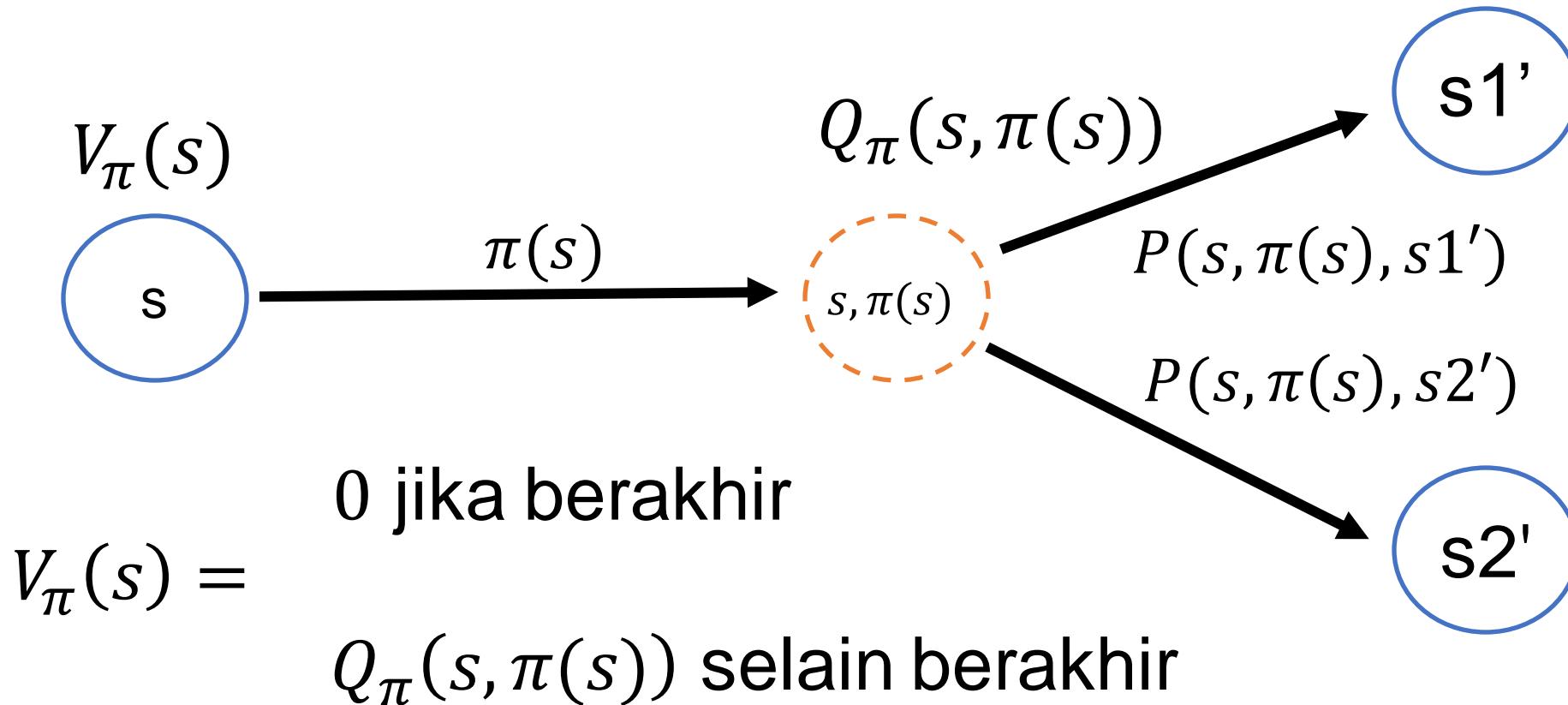
- Policy $\pi(s)$
- Transition Probability M
- Jalur (path)
- Return (Utility) G_t
- Value
- Expected?
- State (s) node
- Action (a)
- State : Action q(s, a) node
- Probability P(s, a, s')
- Reward r(s, a, s')

Evaluasi sebuah Policy

- Definisi: Value dari policy

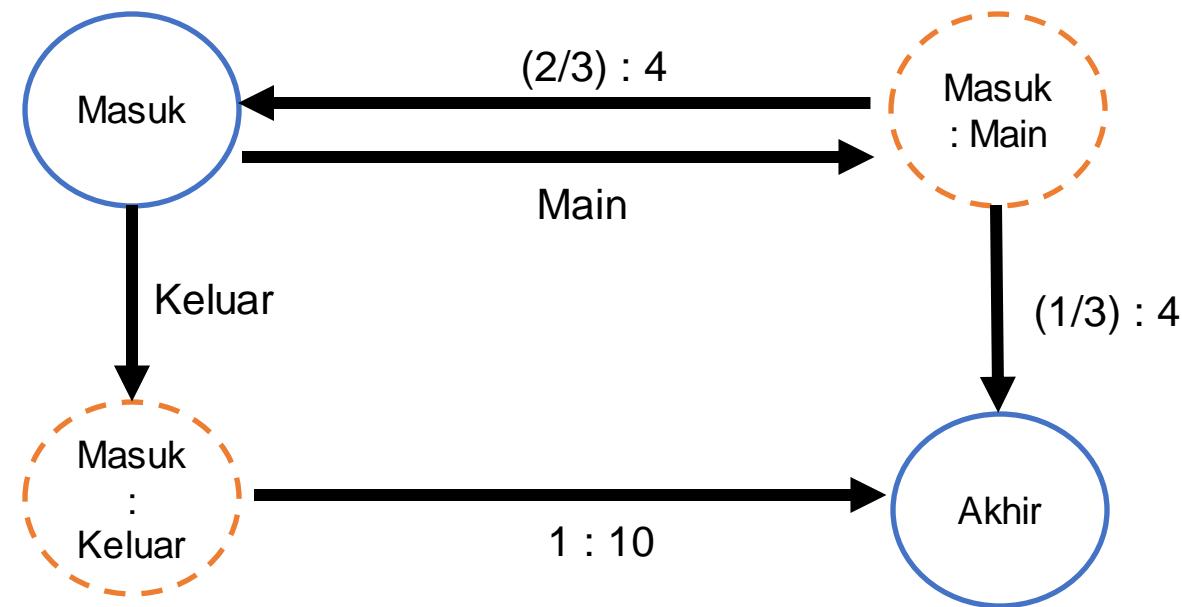


Evaluasi sebuah Policy

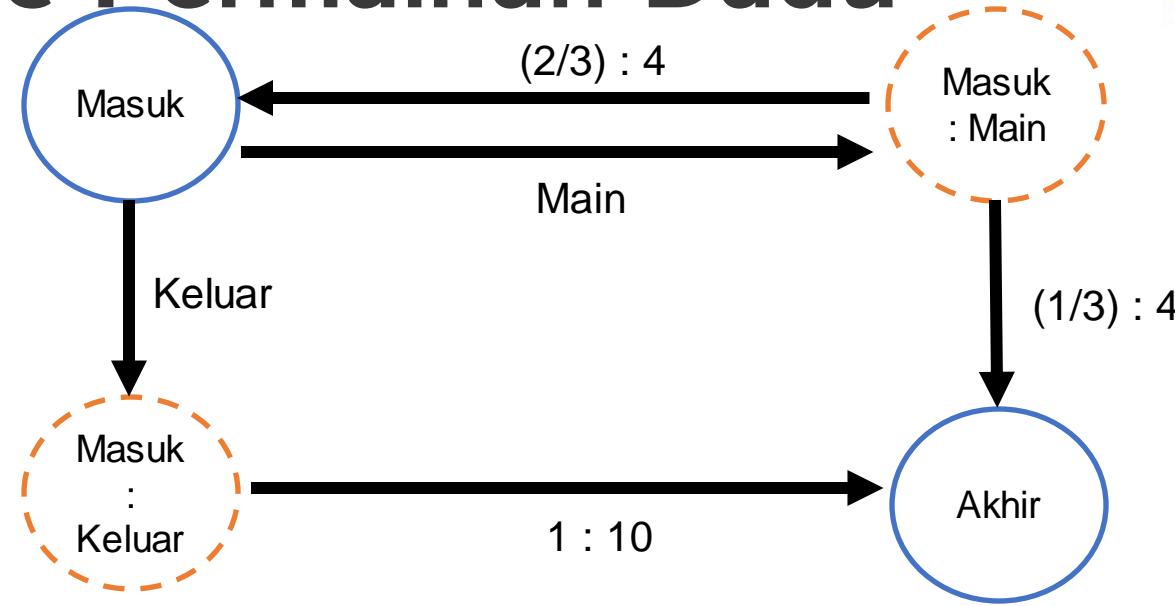


$$Q_{\pi}(s, a) = \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V_{\pi}(s')]$$

Menghitung *value* Permainan Dadu



Menghitung *value* Permainan Dadu



- Misalnya kita pakai policy π “main” maka $\pi(\text{masuk}) = \text{main}$

$$V_{\pi}(\text{masuk}) = \frac{1}{3}(4 + V_{\pi}(\text{akhir})) + \frac{2}{3}(4 + V_{\pi}(\text{masuk}))$$

$$V_{\pi}(\text{akhir}) = 0$$

- Bisa kita selesaikan dengan policy π “main” Value adalah:
(hitung dengan analitik)

$$V_{\pi}(\text{masuk}) = 12$$

Value

- **Value** adalah harapan dari *return* (*Expected return*)

$$Q_{\pi}(s, a) = \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V_{\pi}(s')]$$

- **Value** dari *policy* adalah harapan dari *return* (*Expected return*) ketika mengikuti suatu *policy*

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

Recap and Question?

- Policy $\pi(s)$
- Transition Probability M
- Jalur (path)
- Return (Utility) G_t
- **Value** $V_\pi(s)$
- **Expected** $\mathbb{E}[G_t | S_t = s]$
- State (s) node
- Action (a)
- State : Action q(s, a) node
- Probability P(s, a, s')
- Reward r(s, a, s')

Menghitung value secara iterasi

- Algoritme menggunakan iterasi:
- Inisialisasi $V_{\pi}^{(0)}(s) \leftarrow 0$ untuk semua state
- Untuk interasi $t = 1, \dots, t_{end}$:
 - Untuk setiap state s :

$$V_{\pi}^{(t)}(s) \leftarrow \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_{\pi}^{(t-1)}(s')}$$

Evaluasi Policy (contoh main dadu)

- $V_{\pi}^{(t)}(s) \leftarrow \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_{\pi}^{(t-1)}(s')}$
- $V_{\pi}(masuk) = \frac{1}{3}(4 + V_{\pi}(akhir)) + \frac{2}{3}(4 + V_{\pi}(masuk))$
- Untuk $\pi(masuk) = main$

$\pi(masuk)$ $= main$	$V_{\pi}^{(0)}$	$V_{\pi}^{(1)}$	$V_{\pi}^{(2)}$	$V_{\pi}^{(3)}$	$V_{\pi}^{(4)}$	$V_{\pi}^{(5)}$	$V_{\pi}^{(6)}$
$V_{\pi}(akhir)$	0	0	0	0	0	0	0
$\gamma_{\pi}(masuk)$	0	4	6.67	8.44	9.6296	10.419	10.947

Kapan iterasi selesai?

- Saat Value iterasi t dikurangi Value Iterasi t-1 untuk semua state kurang dari konstanta ϵ . secara matematik dapat dituliskan demikian:

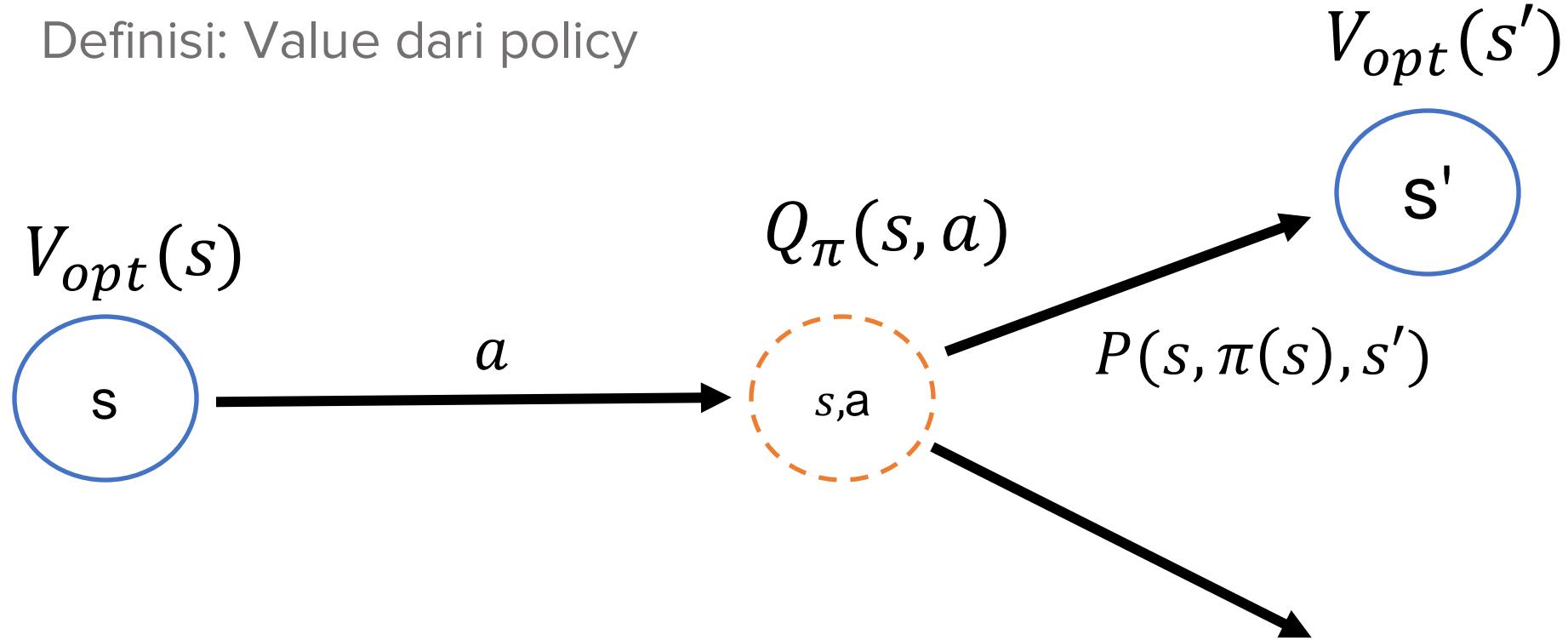
$$\max_{s \in States} |V_\pi^{(t)}(s) - V_\pi^{(t-1)}(s)| \leq \epsilon$$

- Tidak perlu menyimpan semua nilai $V_\pi^{(t)}$ cukup simpan nilai t dan t-1 :

$$V_\pi^{(t)} \text{ dan } V_\pi^{(t-1)}$$

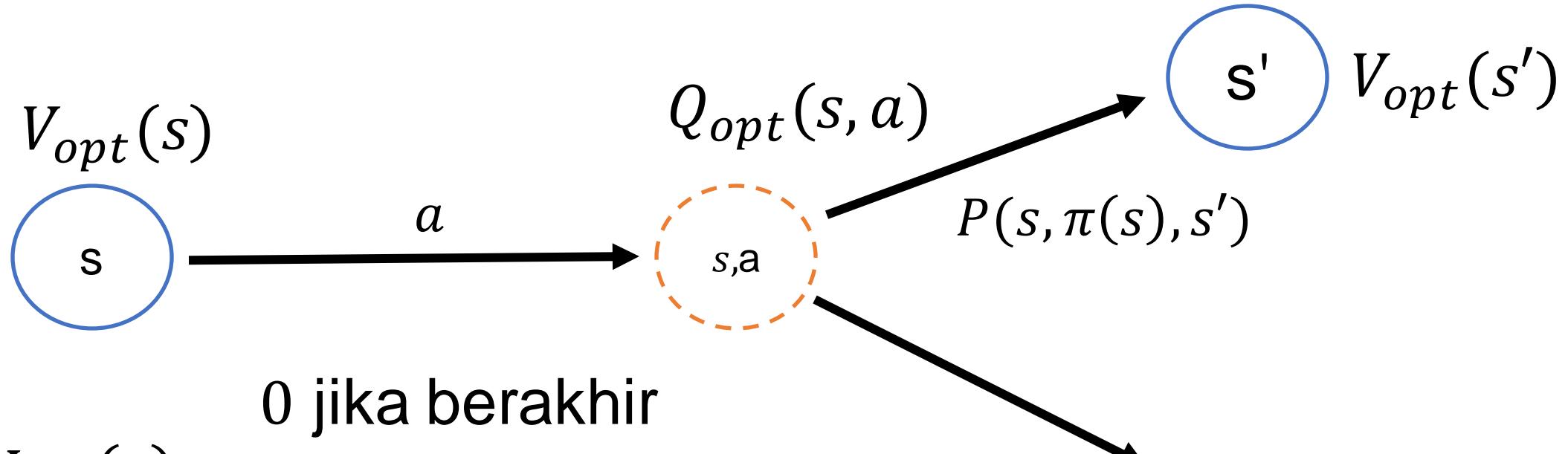
Value dan policy yang Optimal

- Definisi: Value dari policy



$$Q_{opt}(s, a) = \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V_{opt}(s')]$$

Value dan policy yang Optimal



$$V_{opt}(s) = \max_{a \in A(s)} Q_{opt}(s, a) \text{ selain berakhir}$$

$$Q_{opt}(s, a) = \sum_{s'} P(s, a, s') [R(s, a, s') + \gamma V_{opt}(s')]$$

Value dan policy yang Optimal

- $V_{opt}(s) = \max_{a \in A(s)} Q_{opt}(s, a)$

Ya ini adalah Dynamic Programming

- Dynamic Programming jenis **Value Iteration**
- Algoritme menggunakan iterasi:
- Inisialisasi $V_{opt}^{(0)}(s) \leftarrow 0$ untuk semua *state*
- Untuk interasi $t = 1, \dots, t_{end}$:
 - Untuk setiap state s :
$$V_{opt}^{(t)}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_{\pi}^{(t-1)}(s')]$$
 - Kompleksitas adalah $O(t_{end}SAS')$

Konvergensi

- Bagaimana agar Value iteration agar mencapai *konvergensi*?
 - Diskon $\gamma < 1$ atau
 - Grafik MDP yang tidak berbentuk siklus
- Maka value Iteration akan mencapai konvergensi.

Kesimpulan Algoritma Hari ini:

- Evaluasi Policy itu adalah $(MDP, \pi) \rightarrow V_\pi$
- Value Iteration itu adalah $MDP \rightarrow V_{opt}, \pi_{opt}$



TERIMA KASIH

Orbit Future Academy

PT Orbit Ventura Indonesia
Center of Excellence (Jakarta Selatan)
Gedung Veteran RI, Lt.15
Unit Z15-002, Plaza Semanggi
Jl. Jenderal Sudirman Kav.50, Jakarta
12930, Indonesia

- Jakarta Selatan/Pusat
- Jakarta Barat/BSD
- Kota Bandung
- Kab. Bandung
- Jawa Barat

Hubungi Kami

Director of Sales & Partnership
ira@orbitventura.com
+62 858-9187-7388

Social Media

-  Orbit Future Academy
-  @OrbitFutureAcademyIn1
-  OrbitFutureAcademy
-  Orbit Future Academy

Welcome Everyone

AI Mastery Course



Modul Domain AI

Reinforcement Learning

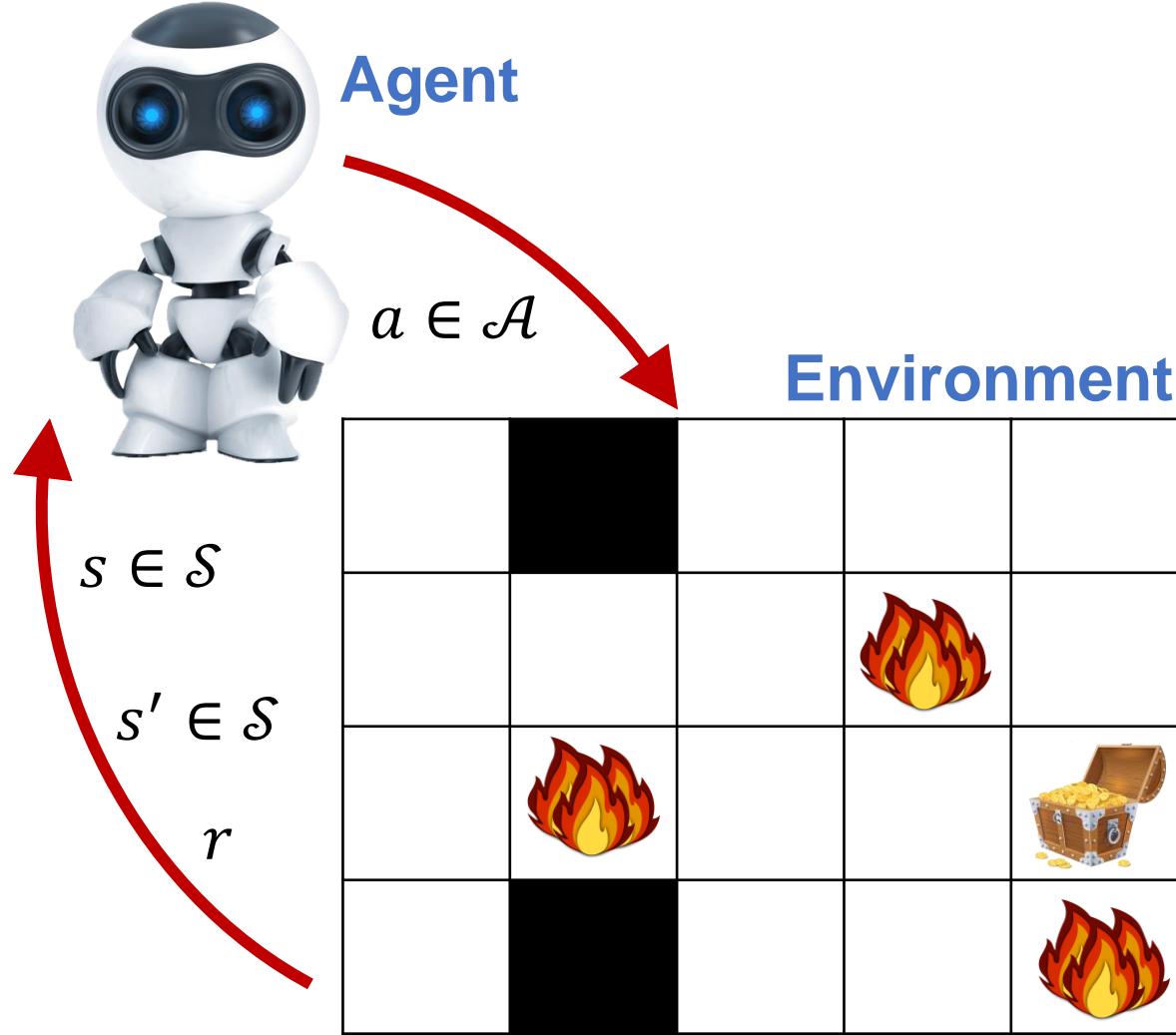
Bagian

Monte Carlo Prediction



Materi sebelumnya: MDP (*the concept*)

Markov Decision Process (S, A, P, R, γ)



$$P = \begin{bmatrix} s'_1 & s'_2 & \dots & s'_N \\ p_{11} & p_{12} & \dots & p_{1N} \\ p_{21} & p_{22} & \dots & \dots \\ \vdots & \vdots & \ddots & \vdots \\ p_{N1} & \dots & \dots & p_{NN} \end{bmatrix}$$

Materi sebelumnya: DP (*the algorithm*)

Algoritme Policy Iteration

$$V_{t+1}(s) = \sum_a \pi(a|s) \sum_{s',r} P(s',r|s,a)[r + \gamma V_t(s')] \dots \dots \dots \text{(Eq. 1)}$$

Materi sebelumnya: DP (*the algorithm*)

Algoritme Value Iteration

- 1: Inisialisasi V_0
 - 2: **Ulangi**
 - 3: Improve V_{t+1} menggunakan estimasi dari V_t (Eq. 3)
(Persamaan 3)
 - 4: **Sampai** konvergen
-

$$V_{t+1}(s) = \max_a \sum_{s',r} P(s',r|s,a)[r + \gamma V_t(s')] \quad \dots\dots\dots \text{(Eq. 3)}$$

Materi Sebelumnya: DP (*the code*)



Frozen Lake Problem

Agent berjalan menuju tujuan yang jalurnya berupa es. Namun ada beberapa lubang. Es-nya licin, sehingga agent tidak bisa selalu bergerak sesuai arah yang diinginkannya.

Materi Sebelumnya: DP (*the code*)

Frozen Lake Problem

- Satu episode berhenti ketika agent mencapai tujuan (goal).
- Agent mendapat 1 reward ketika mencapai tujuan (goal) dan 0 untuk kondisi yang lain.
- Jika masuk ke lubang, agent hanya bisa bergerak searah dengan arah sebelum masuk lubang (peluang $1/3$), dan kedua arah tegak lurus (masing-masing peluangnya $1/3$).



S = starting point

F = frozen surface

H = hole

G = goal

Aplikasi DP pada Permasalahan Riil

Untuk mengaplikasikan algoritme DP, dibutuhkan pengetahuan tentang model environment.



Model tersebut adalah yang terkait matriks probabilitas transisi dan sistem reward-nya. ***DP = Model-based algorithm.***

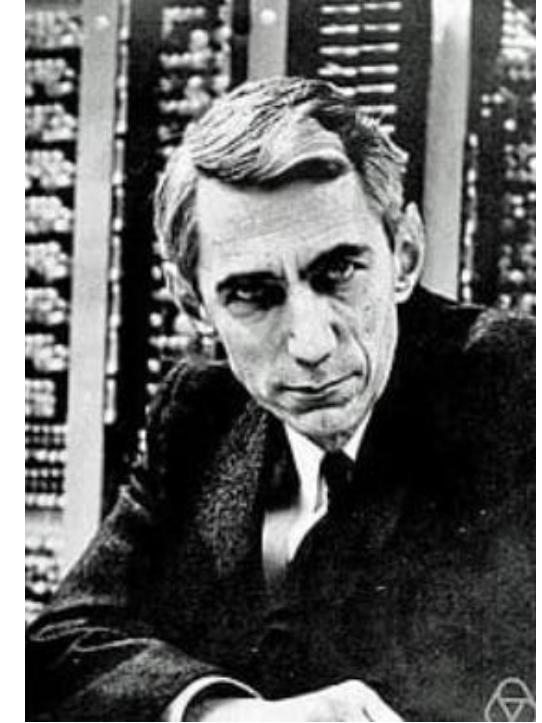
Meskipun matriks probabilitas transisi dapat ditentukan dari pengalaman, butuh sumber daya (waktu, tenaga, uang, dll) yang besar untuk mendapatkan pengalaman yang cukup.

Aplikasi DP pada Permasalahan Rii



Pada tahun 1950, ahli matematika bernama Claude Shannon menulis artikel tentang “*How to Programme a Computer for Playing Chess*”.

Dalam artikel tersebut, dituliskan bahwa jumlah pergerakan yang mungkin untuk rata-rata permainan catur diestimasi sebesar 1×10^{120} langkah.



Bisakah environment catur tersebut dimodelkan? **Bisa.**

Efektifkah untuk dilakukan? **Untuk banyak kasus, tidak efektif.**

Apakah ada solusi mengimplementasikan RL tanpa memodelkan environment-nya secara lengkap? **Ada. Model free algorithm → Monte Carlo.**

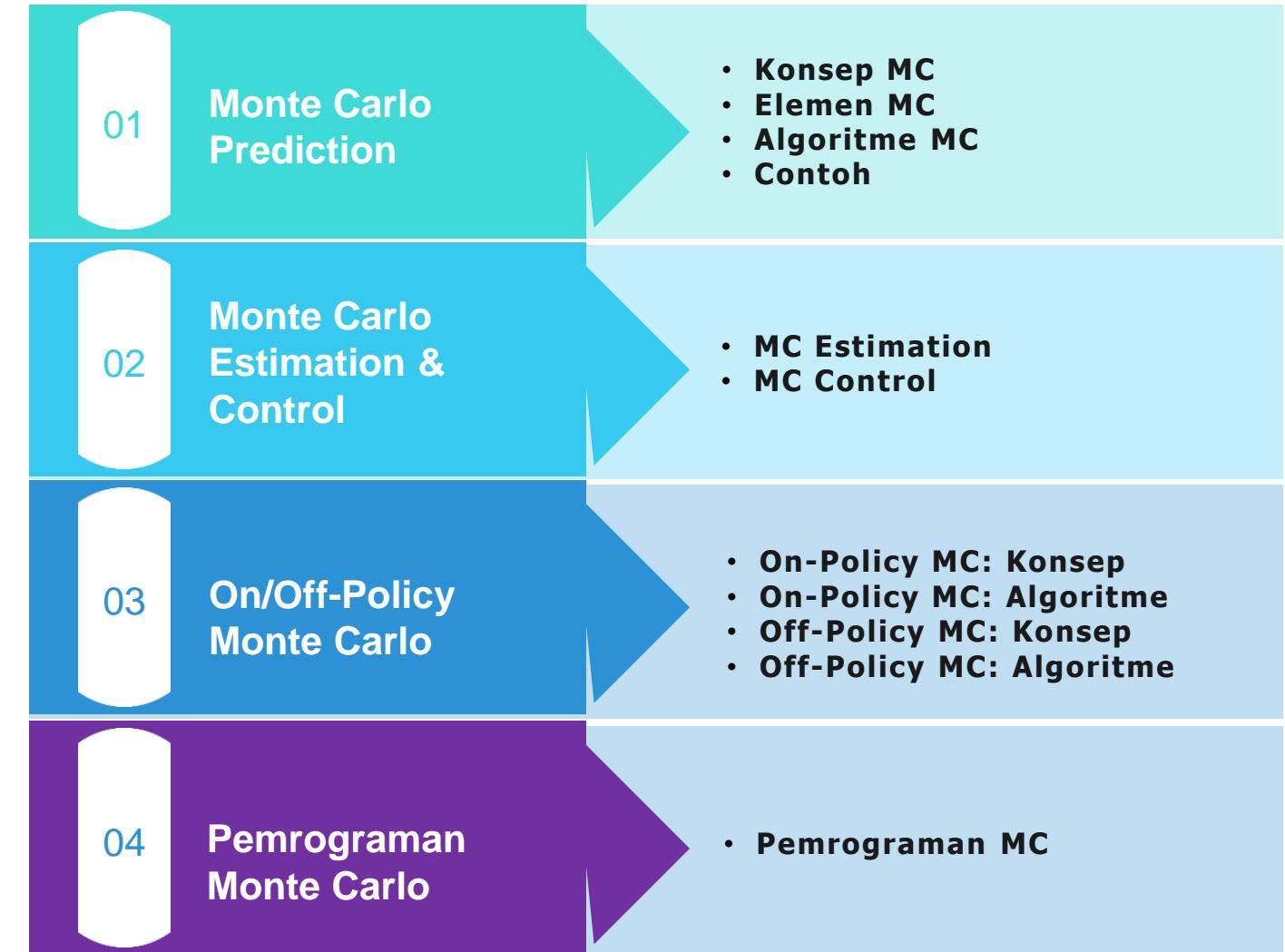


Learning Objectives

- Mengetahui Monte Carlo Prediction
- Mengetahui Monte Carlo Estimation & Control
- Mengetahui On-Policy dan Off-Policy Monte Carlo
- Mengetahui Dasar Pemrograman Monte Carlo



Agenda





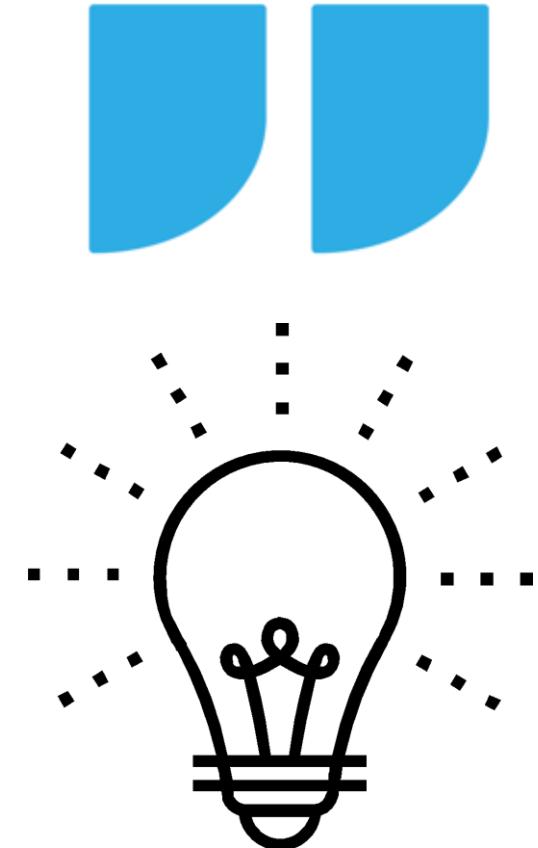
01

Monte Carlo Prediction

- Konsep MC
- Elemen MC
- Algoritme MC
- Contoh

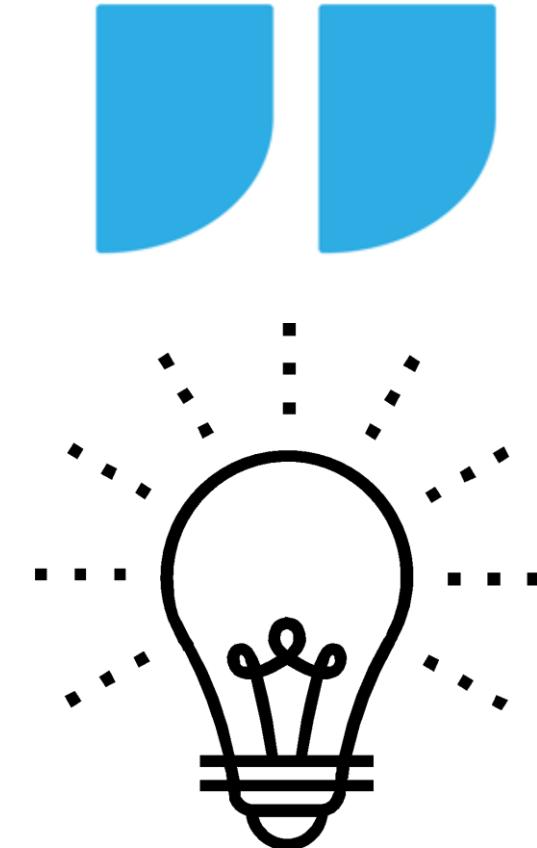
Konsep Monte Carlo (MC)

- MC tidak mengambil pengetahuan lengkap dari environment.
- MC belajar dari experience, episode per episode, baik itu experience aktual, maupun simulasi.
- MC belajar dari episode-episode secara utuh dan independent, tidak bootstrapping.



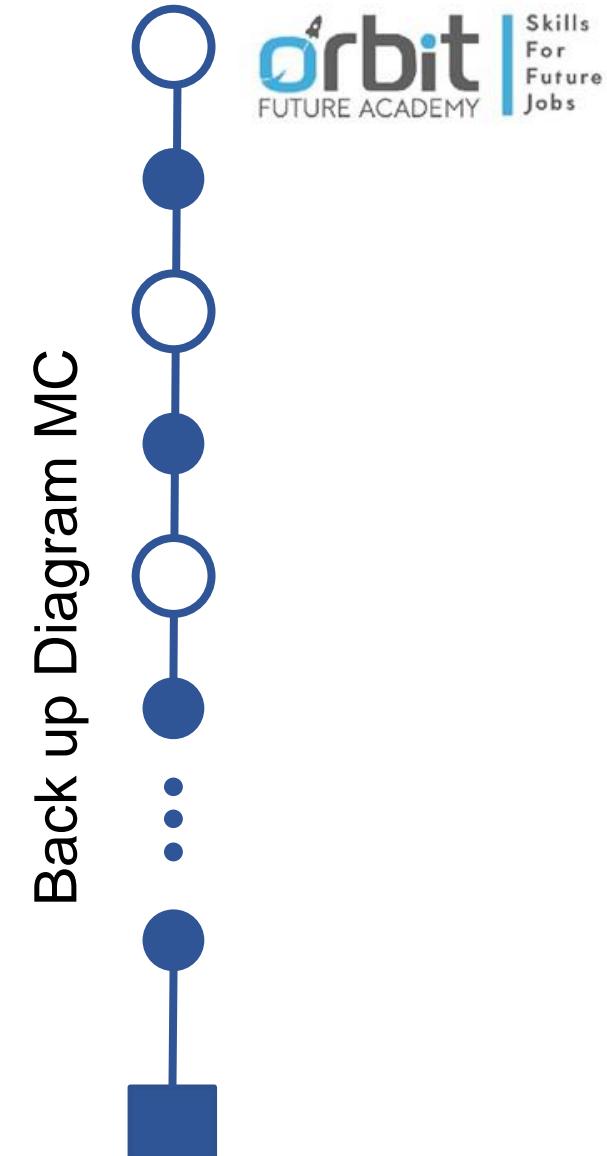
Konsep Monte Carlo (MC) cont'd

- MC didefinisikan untuk jenis episodic environment.
- Ide utama dari MC: Value didapatkan dari rata-rata returns.
- Dengan semakin banyak returns, nilai rata-ratanya diharapkan konvergen pada expected value.



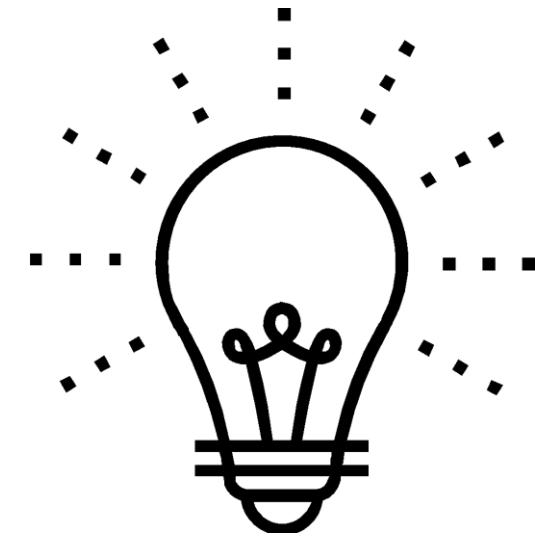
Konsep Monte Carlo (MC) cont'd

- Seluruh episode dipertimbangkan dalam MC
- Hanya satu pilihan setiap perpindahan state di MC, sedangkan DP mempertimbangkan semua probabilitas transisi pada setiap perpindahan state
- Estimasi-estimasi untuk semua state adalah independent di MC, tidak bootstrap
- Waktu yang dibutuhkan untuk mengestimasi suatu state tidak bergantung pada jumlah total state



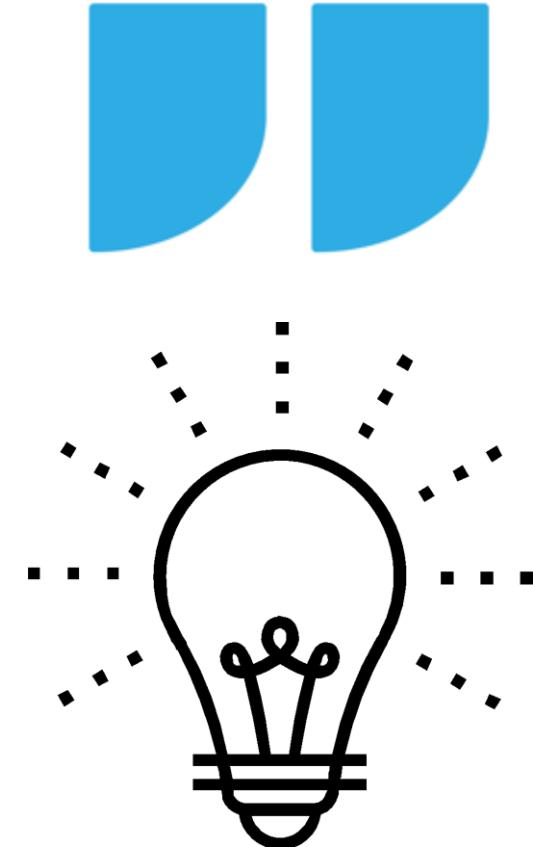
Quiz 1

- Sebutkan perbedaan karakteristik DP dan MC!



Quiz 2

- Mengapa penghitungan nilai rata-ratanya dari returns bisa menyebabkan iterasi MC konvergen pada expected value?



Elemen Algoritme MC

- **Goal** :: Belajar v_π melalui episode-episode dari experience yang patuh pada policy π :

$$S_1, A_1, R_2, \dots, S_t \sim \pi$$

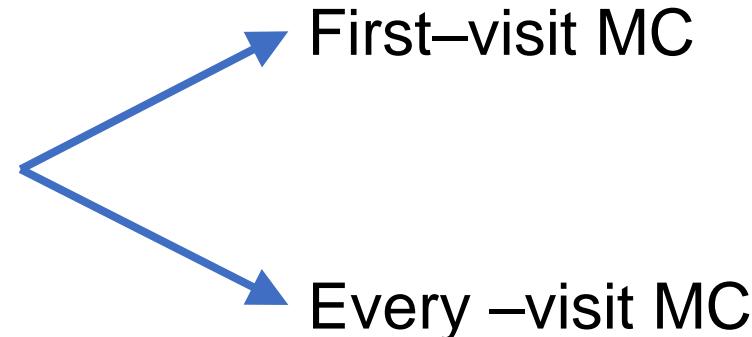
- **Return** :: Adalah total discounted reward :

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- **Value Function** :: Adalah expected return :

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

Rata-rata
(mean)



Elemen Algoritme MC cont'd

Perhitungan Value Function di MC:

- Total counter $N(s) \leftarrow N(s) + 1$
- Total return $S(s) \leftarrow S(s) + G_t$
- Value $V(s) = S(s) / N(s)$
- $V(s)$ konvergen ke v_π sejauh $N(s)$ mendekati ∞



$N(s)$

First-visit MC :: $N(s)$ dihitung saat pertama kali mengunjungi state s pertama dalam setiap episode.



Every-visit MC :: $N(s)$ dihitung setiap mengunjungi state s dalam setiap episode.

Algoritme MC (First–Visit)

Algoritme First–Visit MC

- 1: **Input:** sebuah policy π untuk dievaluasi
 - 2: **Inisialisasi:**
 - 3: $V(s) \in \mathbb{R}$, untuk semua $s \in \mathcal{S}$
 - 4: $Return(s) \leftarrow$ list kosong, untuk semua $s \in \mathcal{S}$
 - 5: **Sampai** konvergen
 - 6: Generate episode berdasarkan π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
 - 7: $G \leftarrow 0$
 - 8: Loop untuk setiap langkah dari episode, $t = T - 1, T - 2, \dots, 0$:
 - 9: $G \leftarrow \gamma G + R_{t+1}$
 - 10: Jika sampai pada first visit ke s :
 - 11: Append G ke $return(S_t)$
 - 12: $V(S_t) \leftarrow$ rata rata dari $Return(S_t)$
-

Algoritme MC (Every–Visit)

Algoritme Every–Visit MC

- 1: **Input:** sebuah policy π untuk dievaluasi
 - 2: **Inisialisasi:**
 - 3: $V(s) \in \mathbb{R}$, untuk semua $s \in \mathcal{S}$
 - 4: $Return(s) \leftarrow$ list kosong, untuk semua $s \in \mathcal{S}$
 - 5: **Sampai** konvergen
 - 6: Generate episode berdasarkan π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
 - 7: $G \leftarrow 0$
 - 8: Loop untuk setiap langkah dari episode, $t = T - 1, T - 2, \dots, 0$:
 - 9: $G \leftarrow \gamma G + R_{t+1}$
 - 10: Append G ke $Return(S_t)$
 - 11: $V(S_t) \leftarrow$ rata-rata dari $Return(S_t)$
-

Contoh

- Misalkan kita punya sebuah *environment* dengan dua state, yaitu A dan B. Kita telah melakukan observasi selama 2 episode dan menghasilkan urutan seperti ini :

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

- **A+3 → A+2** artinya adalah transition dari state A → A dengan reward = 3 untuk transisi tersebut.
- Kita akan coba mencari $V(A)$ dan $V(B)$ menggunakan Algoritma MC first visit dan juga every visit.

Penyelesaian First Visit : Calculating V(A)

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow \underline{A + 3} \rightarrow B - 3 \rightarrow \text{terminate}$

Seperti yang bisa kita lihat kalau kita mempunyai 2 iterasi (episode) yg berbeda, kita akan menjumlahkan semua reward tiap episode dengan catatan reward akan dihitung setelah *agent* mengunjungi *state A*.

$$\begin{aligned}\text{Sum reward episode 1} &= 3 + 2 + (-4) + 4 + (-3) \\ &= 2\end{aligned}$$

$$\begin{aligned}\text{Sum reward episode 2} &= 3 + (-3) \\ &= 0\end{aligned}$$

Penyelesaian First Visit : Calculating V(A)

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Sum reward episode 1 = 2

Sum reward episode 2 = 0

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yg tadi, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

$$V(A) = (\text{Sum reward episode 1} + \text{Sum reward episode 2}) / \text{Jumlah episode yg ada state A}$$

$$V(A) = (2+0) / 2$$

$$V(A) = 1$$

Penyelesaian First Visit : Calculating V(B)

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Seperti yang bisa kita lihat kalau kita mempunyai 2 iterasi (episode) yg berbeda, kita akan menjumlahkan semua reward tiap episode dengan catatan reward akan dihitung setelah *agent* mengunjungi *state* B.

Sum reward episode 1 =

=

Sum reward episode 2 =

=

Penyelesaian First Visit : Calculating V(B)

$A + 3 \rightarrow A + 2 \rightarrow \underline{B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Seperti yang bisa kita lihat kalau kita mempunya 2 iterasi (episode) yg berbeda, kita akan menjumlahkan semua reward tiap episode dengan catatan reward akan dihitung setelah *agent* mengunjungi *state* B.

$$\begin{aligned} \text{Sum reward episode 1} &= -4 + 4 + (-3) \\ &= -3 \end{aligned}$$

$$\begin{aligned} \text{Sum reward episode 2} &= -2 + (3) + (-3) \\ &= -2 \end{aligned}$$

Penyelesaian First Visit : Calculating $V(B)$

$A + 3 \rightarrow A + 2 \rightarrow \underline{B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Sum reward episode 1 = -3

Sum reward episode 2 = -2

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yg tadi, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

$V(B) = (\text{Sum reward episode 1} + \text{Sum reward episode 2}) / \text{Jumlah Episode yang ada state } B$

$V(B) = \dots\dots$

$V(B) = \dots\dots$

Penyelesaian First Visit : Calculating $V(B)$

$A + 3 \rightarrow A + 2 \rightarrow \underline{B - 4} \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Sum reward episode 1 = -3

Sum reward episode 2 = -2

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yg tadi, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

$V(B) = (\text{Sum reward episode 1} + \text{Sum reward episode 2}) / \text{Jumlah Episode yang ada state } B$

$$V(B) = (-3 + (-2)) / 2$$

$$V(B) = -5/2 = -2.5$$

Penyelesaian Every Visit : Calculating V(A)

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Berbeda dengan first visit, pada every visit kita akan menjumlahkan semua reward tiap episode dengan catatan kumulatif reward akan dihitung setiap agent mengunjungi state A.

Sum reward episode 1 $= (3 + 2 + (-4) + 4 + (-3)) + (2 + (-4) + 4 + (-3)) + (4 + (-3))$
 $= 2 + (-1) + 1$

Sum reward episode 2 $= 3 + (-3)$
 $= 0$

Penyelesaian Every Visit : Calculating V(A)

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

$$\begin{aligned}\text{Sum reward episode 1} &= 2 + (-1) + 1 \\ &= 2\end{aligned}$$

$$\text{Sum reward episode 2} = 0$$

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yg tadi, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

$$V(A) = (\text{Sum reward episode 1} + \text{Sum reward episode 2}) / \text{Jumlah kemunculan state A di semua episode}$$

$$V(A) = (2+0) / 4$$

$$V(A) = 0.5$$

Penyelesaian Every Visit : Calculating V(B)

$A + 3 \rightarrow A + 2 \rightarrow \underline{B - 4 \rightarrow A + 4 \rightarrow B - 3} \rightarrow \text{terminate}$

$\underline{\underline{B - 2 \rightarrow A + 3 \rightarrow B - 3}} \rightarrow \text{terminate}$

Berbeda dengan first visit, pada every visit kita akan menjumlahkan semua reward tiap episode dengan catatan kumulatif reward akan dihitung setiap agent mengunjungi state B.

$$\begin{aligned}\text{Sum reward episode 1} &= (-4) + 4 + (-3) + (-3) \\ &= (-3) + (-3)\end{aligned}$$

$$\begin{aligned}\text{Sum reward episode 2} &= (-2) + 3 + (-3) + (-3) \\ &= (-2) + (-3)\end{aligned}$$

Penyelesaian Every Visit : Calculating V(B)

$A + 3 \rightarrow A + 2 \rightarrow \underline{B - 4} \rightarrow A + 4 \rightarrow \underline{B - 3} \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

Sum reward episode 1 $= (-3) + (-3)$
 $= -6$

Sum reward episode 2 $= (-2) + (-3)$
 $= -5$

Lalu setelah menghitung jumlah reward setiap episode berdasarkan catatan yg tadi, kita akan menjumlahkan hasil sum tersebut kemudian membaginya dengan banyaknya episode.

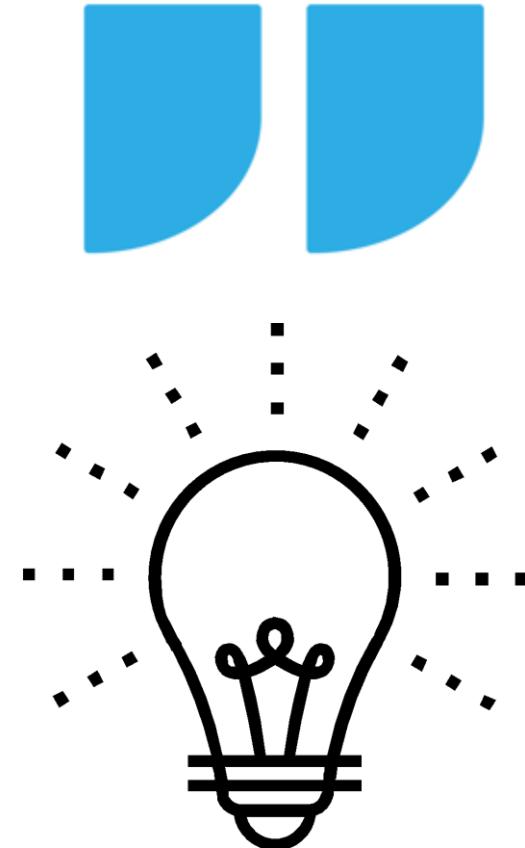
$$V(B) = (\text{Sum reward episode 1} + \text{Sum reward episode 2}) / \text{Jumlah kemunculan state B di semua episode}$$

$$V(B) = ((-6) + (-5)) / 4$$

$$V(B) = -2.75$$

Quiz 3

- Bagaimana perbedaan karakteristik eksplorasi dan eksplorasi dari first-visit MC dengan every-visit MC!





02

Monte Carlo Estimation & Control

- MC Estimation
- MC Control

MC Estimation

- Karena model tidak tersedia, perlu untuk mengestimasi action juga, selain hanya mengestimasi state-nya.
- Tidak seperti DP, pada Model Free Algorithm, MC, state saja tidak cukup untuk menentukan policy.
- Pada MC, action diperlukan dalam menentukan policy. Policy Evaluation Problem.
- Policy evaluation problem mengestimasi $q_{\pi}(s, a)$, yaitu expected return ketika mulai dari state s , melakukan action a , dan mengikuti policy π .



MC Estimation

- First-visit MC mengestimasi value dari pasangan state-action (s, a) dengan merata-rata returns, saat pertama kali menemui (visit) state s dan mengambil action a dalam suatu episode.
- Every-visit MC mengestimasi value dari pasangan state-action (s, a) dengan merata-rata returns, setiap menemui (visit) state s dan mengambil action a dalam suatu episode.
- Implikasinya, tidak semua state-action (s, a) akan ditemui (visited). Apakah masih bisa mendekati expected return, $q_{\pi}(s, a)$?
- Solusi: ***maintaining exploration***



MC Estimation

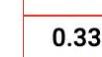
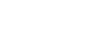
Maintaining Exploration dilakukan dengan:

- Episode-episode memulai dengan sebuah pasangan state-action (s, a) . Mekanisme pertama dari exploring starts.
- Setiap pasangan memiliki probabilitas yang tidak sama dengan nol untuk dipilih sebagai permulaan episode. Mekanisme kedua dari exploring starts.
- Semua pasangan state-action (s, a) memiliki probabilitas tidak sama dengan nol untuk memilih semua action pada setiap state.



MC Estimation

<http://rl-lab.com/gridworld-mc>

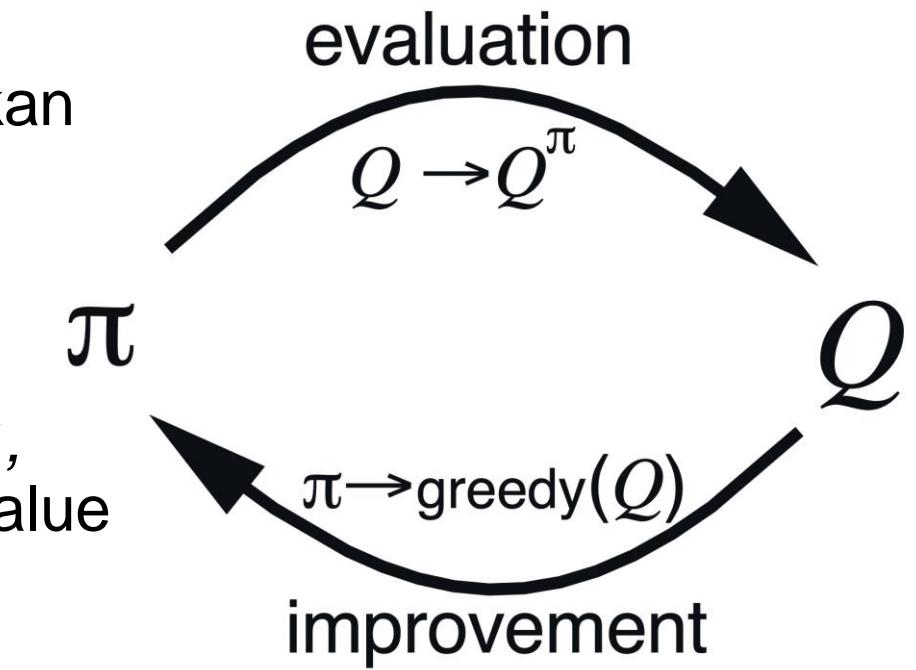
	A	B	C	D	E	F	G	H	I	J
1	 0.18	0.12	0.06	-0.01	0.01	0.00	0.01	0.00	0.00	0.00
2	0.17	0.13	0.05		0.01	0.00	0.00	0.00	0.00	0.00
3	0.33				0.02	0.02	0.00	0.00	0.00	0.00
4	0.41	0.46			0.01	0.01	0.01	0.00	0.00	0.00
5				0.01	0.01	0.00	0.00			
6	0.00	0.00	0.00	0.01	0.00	0.00	0.00		 OUT	0.49
7	0.00	0.00	0.00	0.00	0.00				0.63	0.57
8	0.00	0.00	0.00	0.00	0.00			0.83	0.74	0.64
9	0.00	0.00	0.00	0.00	0.00		0.86	1.00	0.85	0.74
10	0.00	0.00	0.00	0.00	0.00		1.00		1.00	0.85

MC Control

Policy Evaluation: Episode-episode dikerjakan dengan mekanisme exploring starts. Lalu, MC akan menghitung q_{π_k} untuk sembarang π_k

Policy Improvement: Untuk setiap action-value function q , greedy policy-nya, untuk setiap $s \in \mathcal{S}$, memilih sebuah action yang merupakan action-value maksimal, yaitu: $\pi(s) = \arg \max_a q(s, a)$. Policy improvement dilakukan dengan membentuk π_{k+1} sebagai greedy policy berdasarkan pada q_{π_k}

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$





03

On/Off Policy Monte Carlo

- On-Policy MC: Konsep
- On-Policy MC: Algoritme
- Off-Policy MC: Konsep
- Off-Policy MC: Algoritme

On-Policy MC: Konsep

- On-policy MC adalah algoritme Monte Carlo yang melakukan evaluasi atau improvisasi dari policy yang digunakan untuk membuat keputusan-keputusan.
- Pembahasan MC sebelumnya adalah on-policy MC.
- Lalu disini akan dibahas on-policy MC dengan policy dengan nama ε – *greedy*.
- Dalam ε – *greedy* policy, hampir semua action yang dipilih, mempunyai action value estimasi yang maksimal, tetapi dengan probabilitas ε dalam memilih action, tidak dengan random/acak.



On-Policy MC : Konsep

Untuk semua nongreedy action, maka: $\pi(a|S_t) \leftarrow \frac{\varepsilon}{|\mathcal{A}(s)|}$

Yaitu saat $a \neq A^*$



Untuk semua greedy action, maka: $\pi(a|S_t) \leftarrow 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$

Yaitu saat $a = A^*$

Dalam hal ini, jika $\pi(a|S_t) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$

maka policy yang dikerjakan dinamakan $\varepsilon - soft$ policy

On-Policy MC: Algoritme



Algoritme On-Policy First-Visit MC

- 1: **Parameter algoritme:** small $\varepsilon > 0$
 - 2: **Inisialisasi:**
 - 3: $\pi \leftarrow \varepsilon - \text{soft policy}$
 $Q(s, a) \in \mathbb{R}$, untuk semua $s \in \mathcal{S}, a \in \mathcal{A}(s)$
 - 4: $Return(s) \leftarrow \text{list kosong}$, untuk semua $s \in \mathcal{S}, a \in \mathcal{A}(s)$
-

On-Policy MC: Algoritme

Algoritme On-Policy First-Visit MC

- 5: Sampai konvergen (untuk setiap episode)
 - 6: Generate episode berdasarkan $\pi: S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
 - 7: $G \leftarrow 0$
 - 8: Loop untuk setiap langkah dari episode, $t = T - 1, T - 2, \dots, 0$:
 - 9: $G \leftarrow \gamma G + R_{t+1}$
 - 10: Jika sampai pada first visit ke (S_t, A_t) :
 - 11: Append G ke $Return(S_t, A_t)$
 - 12: $Q(s, a) \leftarrow$ rata-rata dari $Return(S_t, A_t)$
 - 13: $A^* \leftarrow \arg \max_a Q(S_t, a)$
 - 14: Untuk semua $a \in \mathcal{A}(S_t)$:
 - 15: $\pi(a|S_t) \leftarrow 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(S)|}$ jika $a = A^*$
 - 16: $\pi(a|S_t) \leftarrow \frac{\varepsilon}{|\mathcal{A}(S)|}$ jika $a \neq A^*$
-

Off-Policy MC: Konsep

- Ada dilemma pada hampir semua metode kontrol. Ingin optimal tetapi juga harus bergerak tidak optimal, yaitu harus eksplorasi.
- Off-policy MC adalah algoritme Monte Carlo yang melakukan evaluasi atau improvisasi dari policy yang berbeda dalam meng-generate data (**behavior policy**), sedangkan **target policy**-nya sama.



Off-Policy MC: Konsep

- Target policy adalah policy yang dipelajari oleh agent. Target policy ini adalah greedy policy yang patuh pada Q.
- Behavior policy (μ) adalah policy yang meng-generate behavior
- Probabilitas dari behavior policy untuk memilih setiap action, yang mungkin dipilih oleh target policy, harus **bukan nol**. Untuk memastikan hal tersebut, kita membutuhkan behavior policy yang **soft**.



Off-Policy MC: Algoritme



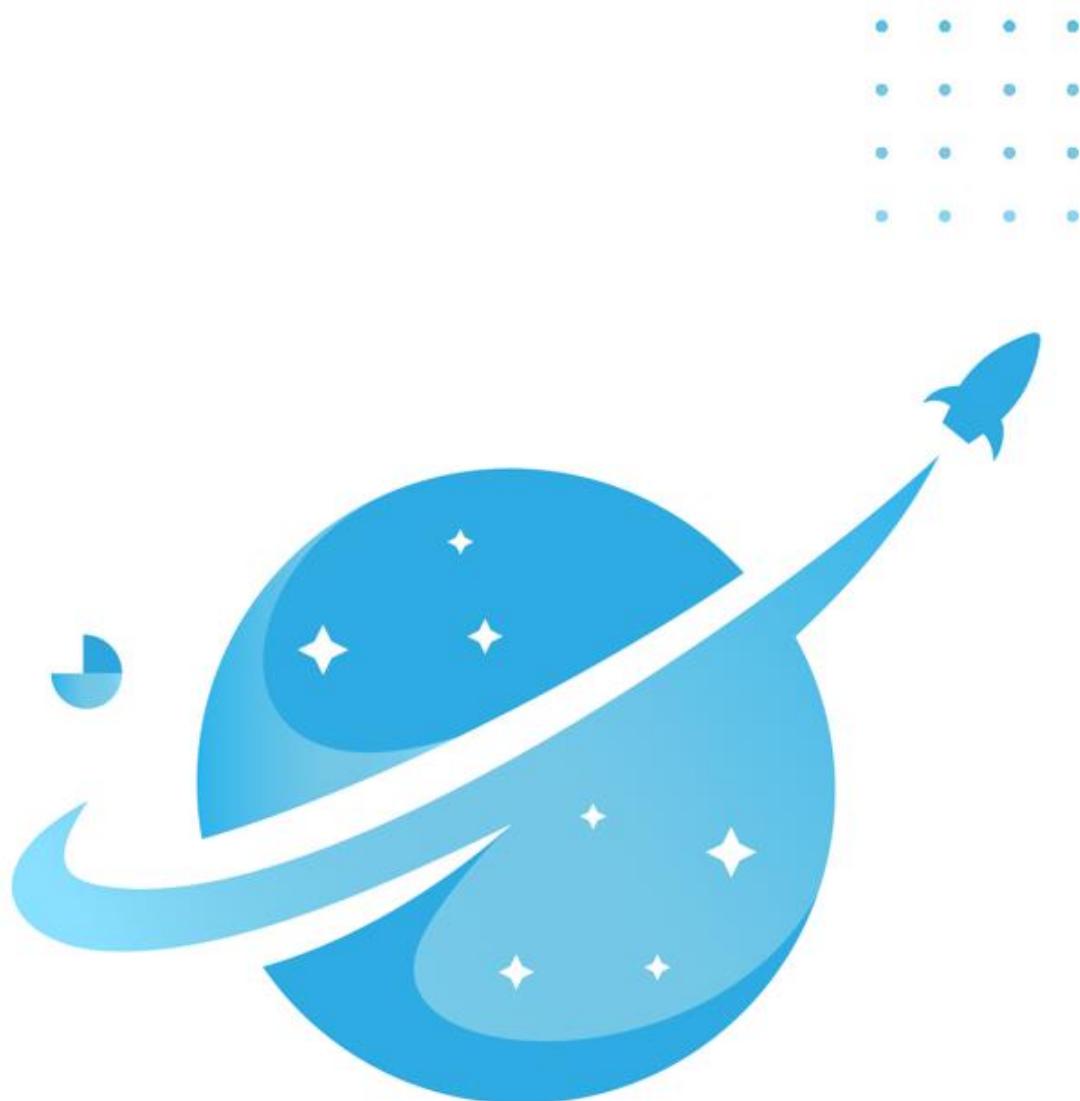
Algoritme Off-Policy First-Visit MC

- 1: **Input:** sebuah policy π untuk dievaluasi
 - 2: **Inisialisasi:**
 - 3: $Q(s, a) \in \mathbb{R}$, untuk semua $s \in \mathcal{S}, a \in \mathcal{A}(s)$
 - 4: $C(s, a) = 0$, untuk semua $s \in \mathcal{S}, a \in \mathcal{A}(s)$, adalah greedy yang patuh pada Q
-

Off-Policy MC: Algoritme

Algoritme Off-Policy Every-Visit MC

- 5: Sampai konvergen (untuk setiap episode)
 - 6: $b \leftarrow$ sembarang policy yang masih dalam lingkup π
 - 7: Generate episode berdasarkan $\pi: S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
 - 8: $G \leftarrow 0$
 - 9: $W \leftarrow 1$
 - 10: Loop untuk setiap langkah dari episode, $t = T - 1, T - 2, \dots, 0$, selama $W \neq 0$:
 - 11: $G \leftarrow \gamma G + R_{t+1}$
 - 12: $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
 - 13: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} (G - Q(S_t, A_t))$
 - 14: $\pi(S_t) = \arg \max_a Q(S_t, a)$
 - 15: Jika $A_t \neq \pi(S_t)$, maka keluar dari loop (lanjutkan episode berikutnya)
 - 16: $W \leftarrow W \frac{1}{\mu(A_t, S_t)}$
-



04

Pemrograman Monte Carlo

- Pemrograman MC

Monte Carlo (*the code*)



Frozen Lake Problem

Bagaimana jika Frozen Lake Problem diselesaikan dengan Monte Carlo?

Summary

1. Monte Carlo bekerja dengan pengalaman dari sample, dan menggunakannya untuk belajar secara langsung tanpa model.
2. Pada first-visit MC, value function dihitung saat pertama kali mengunjungi state s pertama dalam setiap episode.
3. Pada every-visit MC, value function dihitung setiap mengunjungi state s dalam setiap episode.
4. Pada on-policy MC, agent berkomitmen untuk selalu eksplorasi dan mencoba mencari policy terbaik.
5. Pada off-policy MC, dilakukan evaluasi atau improvisasi dari policy yang berbeda dalam meng-generate data (**behavior policy**), sedangkan **target policy**-nya sama.

References

- Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction. 2020
- Sudharsan Ravichandiran. Hands-On Reinforcement
Learning with Python. 2018



Student Activities

Cari satu kasus di environment OpenAI yang bisa diselesaikan dengan Monte Carlo, pelajari programnya.

1. Jelaskan yang anda pahami tentang mekanisme kerja dari program tersebut! Deadline 1 jam sebelum pertemuan ke 4.
2. Susunlah algoritma Monte Carlo dalam bentuk pseudocode untuk penyelesaian masalah tersebut!
(Optional) Deadline 1 jam sebelum pertemuan ke 5.
3. Buatlah program untuk algoritma dari jawaban soal nomor 2! **(Optional)** Deadline 1 jam sebelum pertemuan ke 5.
4. Jelaskan perbedaan antara DP dan MC dalam menyelesaikan Frozen Lake Problem.



This presentation is made by



Wayan Dadang

Wijaya Yudha Atmaja

Dino Febriyanto S.Kom.

Ryan Satria Wijaya S.T.

Oetomo





TERIMA KASIH

Orbit Future Academy

PT Orbit Ventura Indonesia
Center of Excellence (Jakarta Selatan)
Gedung Veteran RI, Lt.15
Unit Z15-002, Plaza Semanggi
Jl. Jenderal Sudirman Kav.50, Jakarta
12930, Indonesia

- Jakarta Selatan/Pusat
- Jakarta Barat/BSD
- Kota Bandung
- Kab. Bandung
- Jawa Barat

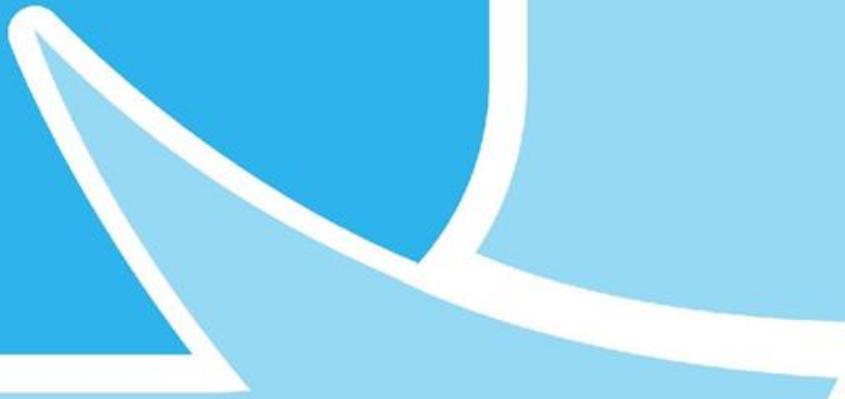
Hubungi Kami

Director of Sales & Partnership
ira@orbitventura.com
+62 858-9187-7388

Social Media

-  Orbit Future Academy
-  @OrbitFutureAcademyIn1
-  OrbitFutureAcademy
-  Orbit Future Academy

AI Mastery Course



Modul Domain AI

Reinforcement Learning
Belajar dengan Penguatan

Bagian

Q Learning dan Deep Q Learning



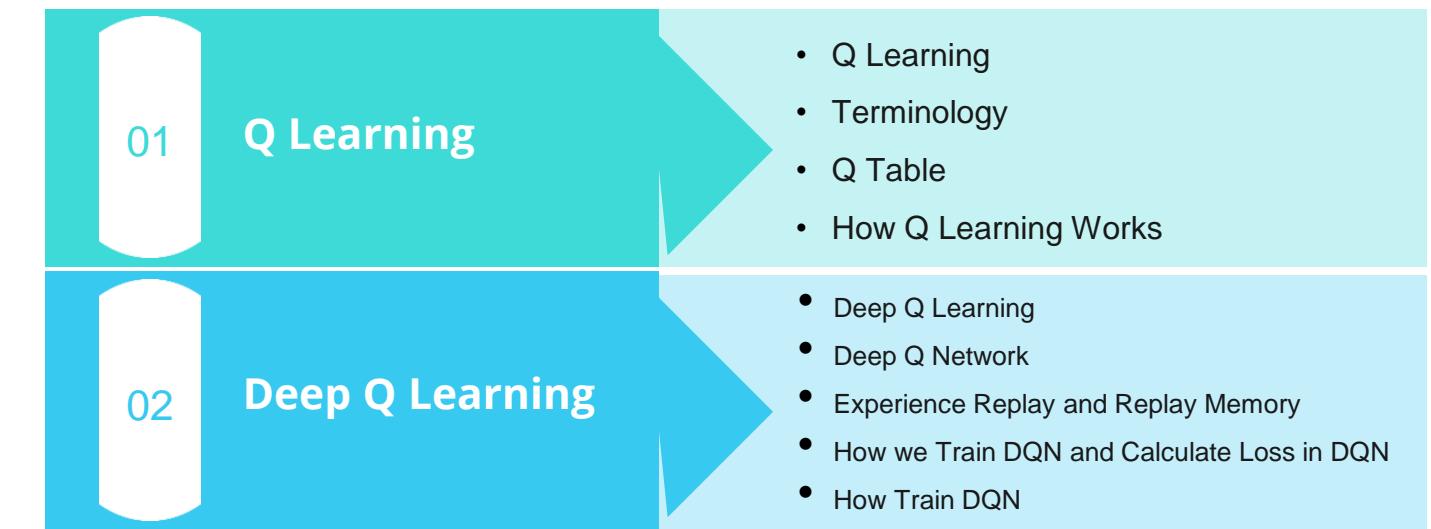


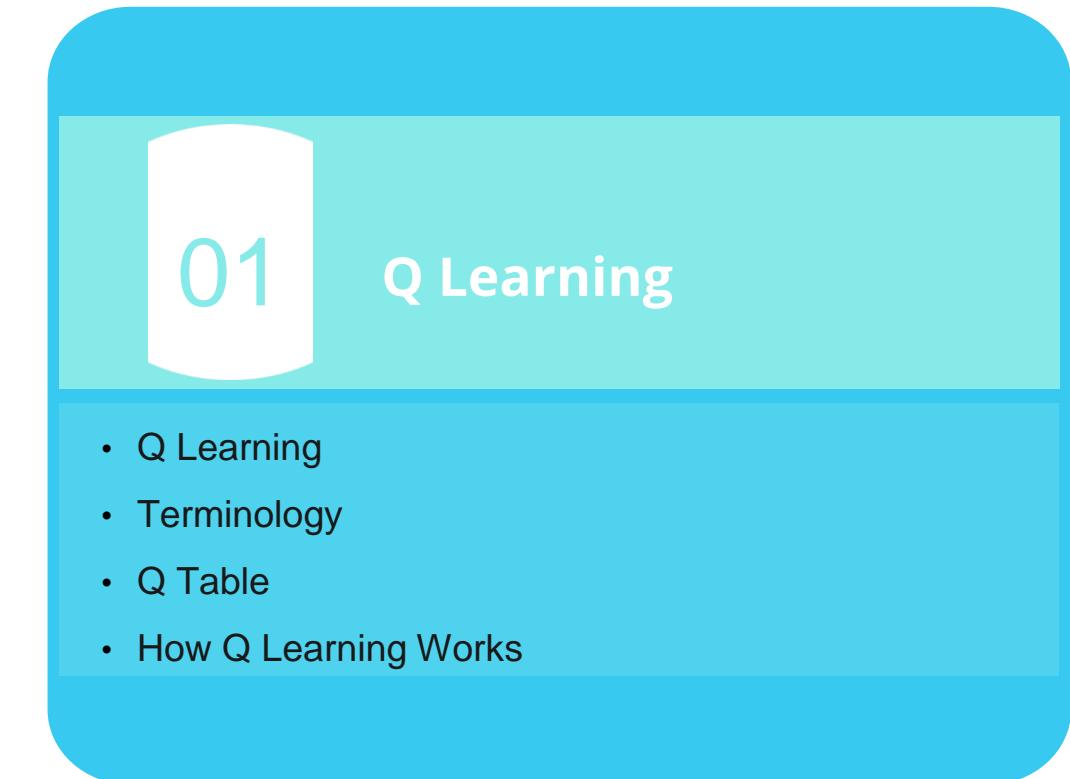
Learning Objectives

1. Mengetahui Q Learning,
2. Memahami bagaimana Q Learning bekerja
3. Mengetahui Deep Q Learning.
4. Memahami bagaimana Deep Q Learning Bekerja



Agenda





01 Q Learning

- Q Learning
- Terminology
- Q Table
- How Q Learning Works

Monte Carlo vs TD Method (Recap)

Monte Carlo update:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$



Monte Carlo vs TD Method (Recap) cont,d

Monte Carlo update:

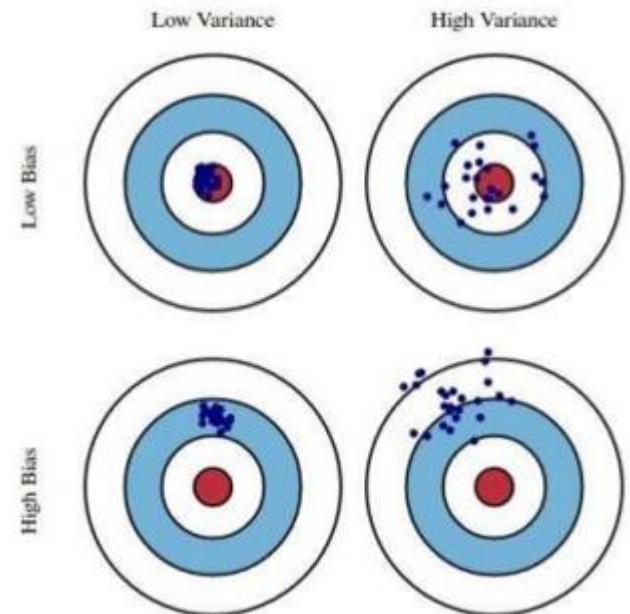
$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- Unbiased

The target is true return of the sample.

- High variance

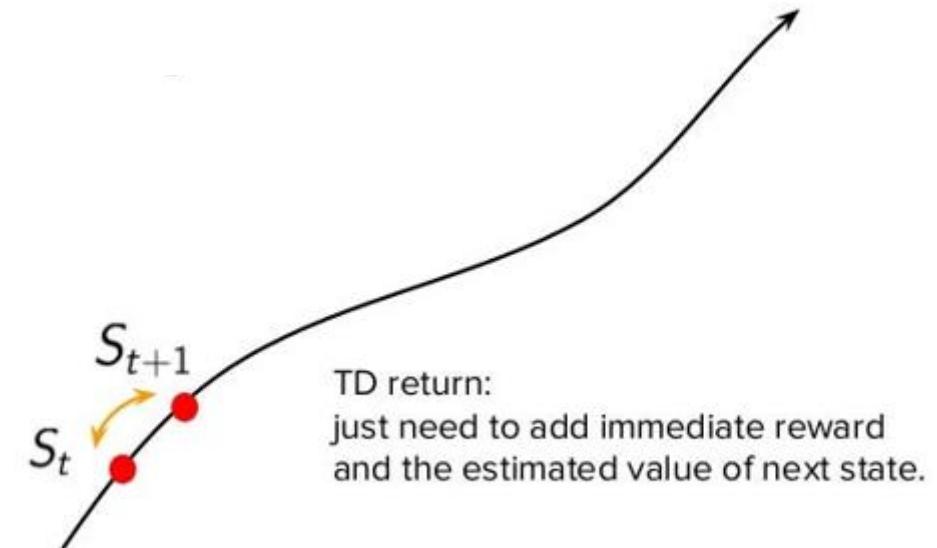
The targets among samples are very different because they are depend on whole trajectory.



Monte Carlo vs TD Method (Recap)

TD method:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]\mathbf{c}$$



Monte Carlo vs TD Method (Recap) cont,d

TD method:

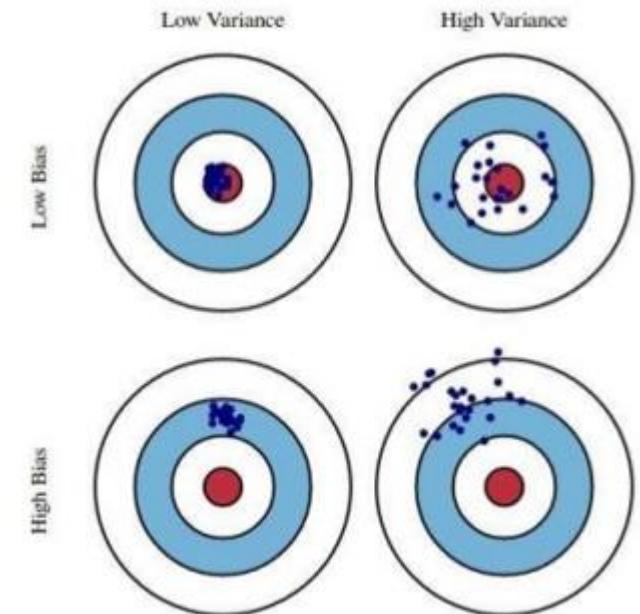
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]c$$

- Biased

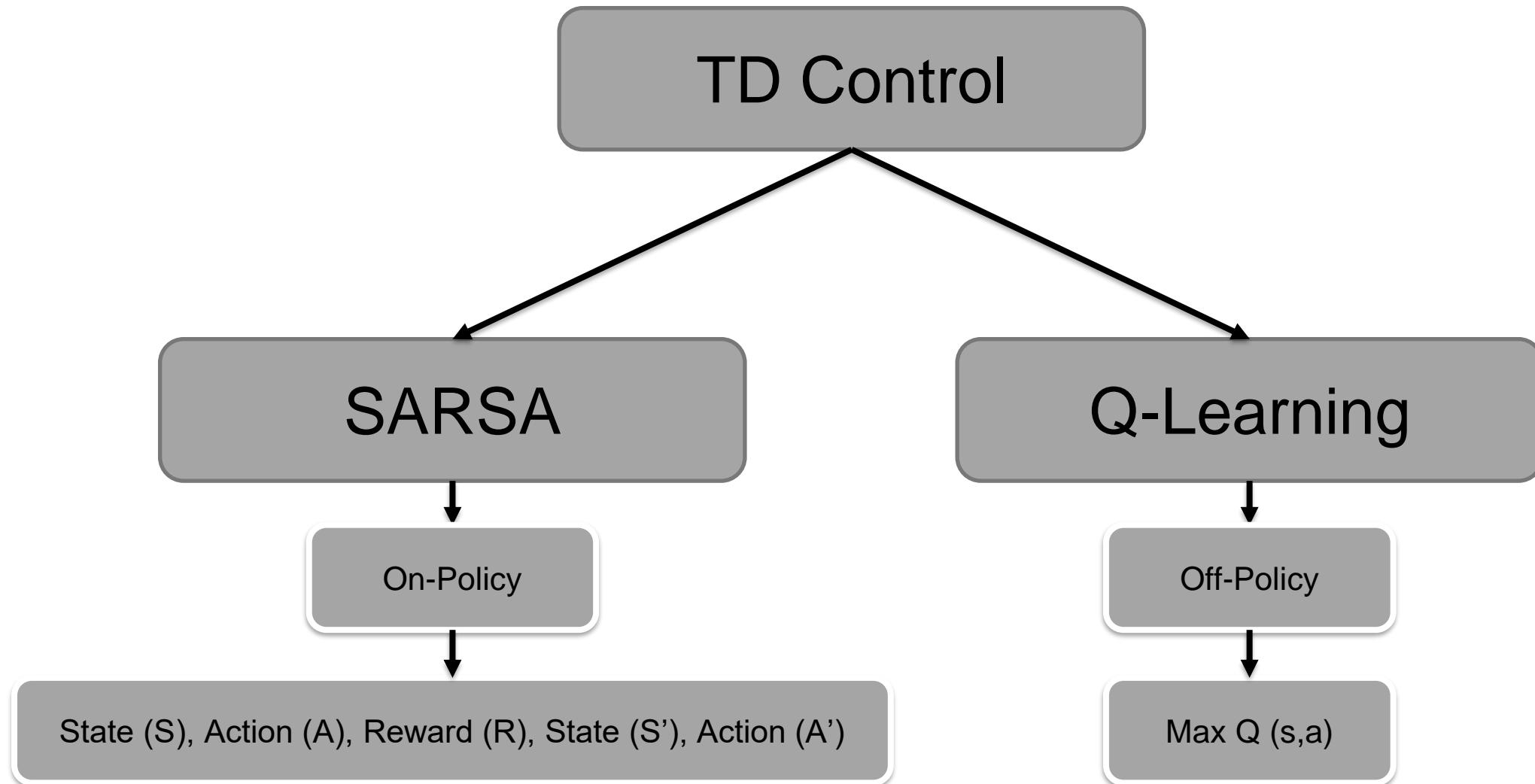
The target is also an estimated value (because of $V(s)$).

- Low variance

The targets are slightly difference, because it just take one-step.



Temporal Difference Learning (Recap)



SARSA VS Q-Learning

- On-policy: Sample policy sama dengan learning policy (target policy)

Contoh: Sarsa dan Policy Gradient

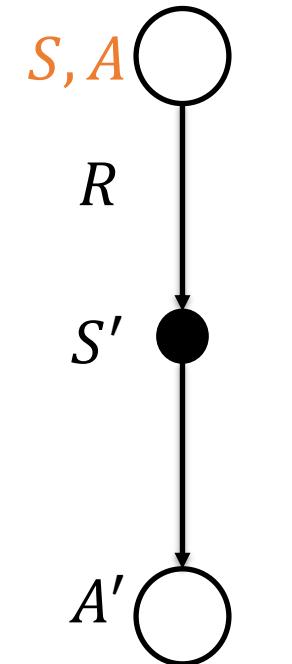
- Off-policy: Sample policy berbeda dengan learning policy (target policy)

Contoh: Q-Learning dan Deep Q Network (DQN)

SARSA – On Policy TD Control

- Terinspirasi dari policy iteration
- Mengganti value function (V_π) dengan action-value function
- On Policy
- Fokus pada state-action (S, A)

$$Q(\underline{S_t, A_t}) \leftarrow Q(\underline{S_t, A_t}) + \alpha [R_{t+1} + \gamma Q(\underline{S_{t+1}, A_{t+1}}) - Q(\underline{S_t, A_t})]$$



SARSA

Q-Learning – Off Policy TD Control

Q-Learning merupakan pengembangan RL yang menggunakan *Q-values* (disebut juga *action-values*) untuk meningkatkan kemampuan *agent* belajar *agent* berulang-ulang.

Konsep dasar *Q-Learning*:

- Terinspirasi dari *value iteration*
- *Sample an action*
- *Observe the reward and the next state*
- *Take the action with the highest Q (Max Q)*

Action dari setiap step dapat dihitung untuk menemukan *action* terbaik (*best action*). Untuk keperluan ini digunakan *Q-Table*.

Q-Learning – Off Policy TD Control

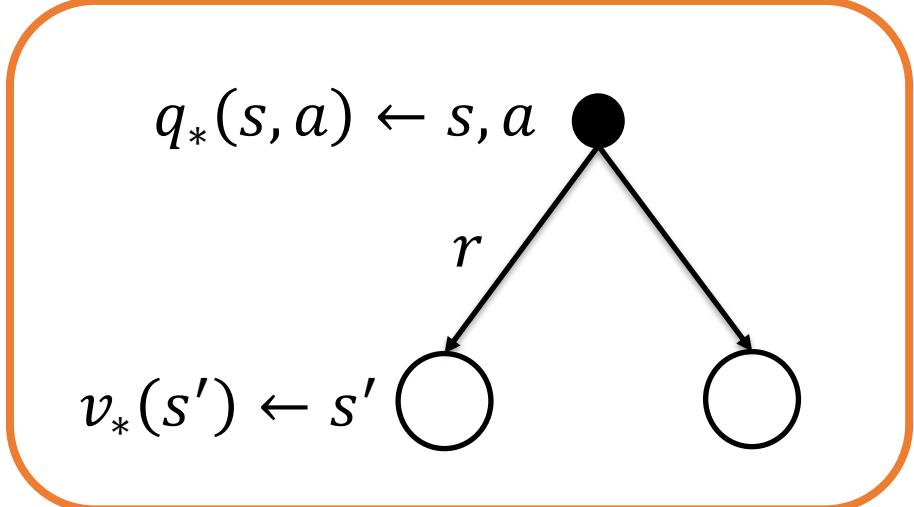
- Terinspirasi dari value iteration

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

$$q_*(s, a) \leftarrow s, a$$

r

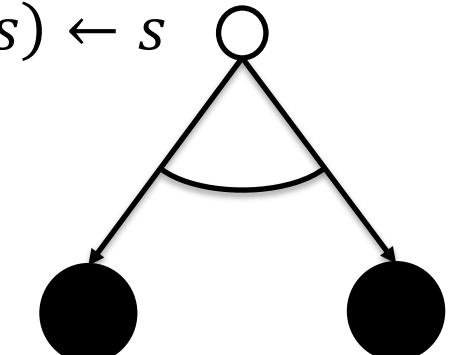
$$v_*(s') \leftarrow s'$$



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

$$v_*(s) \leftarrow s$$

$$q_*(s, a) \leftarrow a$$



Q-Learning – Off Policy TD Control

- Terinspirasi dari value iteration

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

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

Q-Learning – Off Policy TD Control

- Terinspirasi dari value iteration

$$Q(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') \right]$$

- Q-Learning update

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Algoritma Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Parameter algoritma: step size $\alpha \in (0,1)$, small $\varepsilon > 0$

Inisialisasi $Q(s, a)$, untuk semua $s = S^+, a \in \mathcal{A}(s)$, arbitrary except that $Q(\text{terminal}, \cdot) = 0$

Loop untuk setiap episode:

 Inisialisasi S

 Loop untuk setiap step dari episode:

 Pilih A dari S menggunakan policy derived from Q (e.g., ε -greedy)

 Ambil action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right]$$

$S \leftarrow S'$

 hingga S is terminal

Algoritma Q-Learning secara sederhana:

1. Tentukan current state = initial state.
2. Dari current state, cari dengan nilai Q terbesar.
3. Tentukan current state = next state.
4. Ulang Langkah (2) dan (3) hingga current state = goal state.

Algoritma Q-Learning (pseudocode):

Algorithm :Q-learning – Learn function

Require:

States $X = \{1, \dots, n_x\}$

Action $\mathcal{A} = \{1, \dots, n_a\}, A: X \Rightarrow \mathcal{A}$

Reward function $R: X \times \mathcal{A} \rightarrow \mathbb{R}$

Black – box (probabilistic) transition function $T: X \times \mathcal{A} \rightarrow X$

Learning rate $\alpha \in [0,1], \alpha = 0.1$

Discounting factor $\gamma \in [0,1]$

Procedure $Q - Learning(X, \mathcal{A}, R, T, \alpha, \gamma)$

Inisialisasi $Q: X \times \mathcal{A} \rightarrow \mathbb{R}$ arbitrarily

while Q is not converged do

Start in state $s \in X$

while s is not terminal do

Calculate π according to Q and exploration strategy (e. g. $\pi(x) \leftarrow \arg \max_a Q(x, a)$)

$a \leftarrow \pi(s)$

$r \leftarrow R(s, a)$

$s' \leftarrow T(s, a)$

$Q(s', a) \leftarrow (1 - \alpha).Q(s, a) + \alpha.(r + \gamma \cdot \max_{a'} Q(s', a'))$

$s \leftarrow s'$

end

end

return Q

Q-Learning

Objektif dari Q Learning adalah mencari *policy* yang optimal berdasarkan in the sense that the expected value of the total reward over all successive steps is the maximum achievable. Atau dengan kata lain, tujuan dari Q-Learning adalah mencari *policy* yang optimal dan mencari optimal Q-Values untuk setiap pasangan state-action.

Terminology

Policy (π) : Adalah sebuah fungsi yg memetakan setiap *state* ke *action*. Sehingga dengan mengikuti policy, agent bisa memilih *action* apa yg akan dia ambil pada *state* tertentu.

Reward (r) : *feedback* atau umpan balik untuk agent. Reward dapat bernilai positif (berupa hadiah) atau negatif (berupa hukuman) dan juga nol (tidak ada tindakan apapun terhadap agent).

Episodes: Ketika *agent* berakhir pada terminating state dan tidak bisa mengambil *action* apapun pada proses learning.

Terminology (cont)

Q Function : adalah fungsi yang digunakan dalam Q-Learning untuk mencari Q-Value untuk setiap pasangan *state* dan *action* dan untuk menentukan apakah sebuah *action* akan baik jika dilakukan pada *state* tertentu.

Lalu bagaimana kita menemukan Q-Value yang terbaik menggunakan Q-Function? Kita gunakan persamaan berikut :

$$q_*(s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') \right]$$

Terminology (cont)

Q-Table : adalah sebuah table yang digunakan untuk menyimpan Q-Value untuk setiap pasangan *state* dan *action*. Horizontal axis dari table ini merepresentasikan kumpulan *action*, dan *vertical axis* merepresentasikan kumpulan *state* yang ada. Jadi, dimensi dari table akan bergantung terhadap jumlah *actions* dan jumlah *states*.

Q table

	1	2	3	4
1				
2		 		
3				



States

Actions

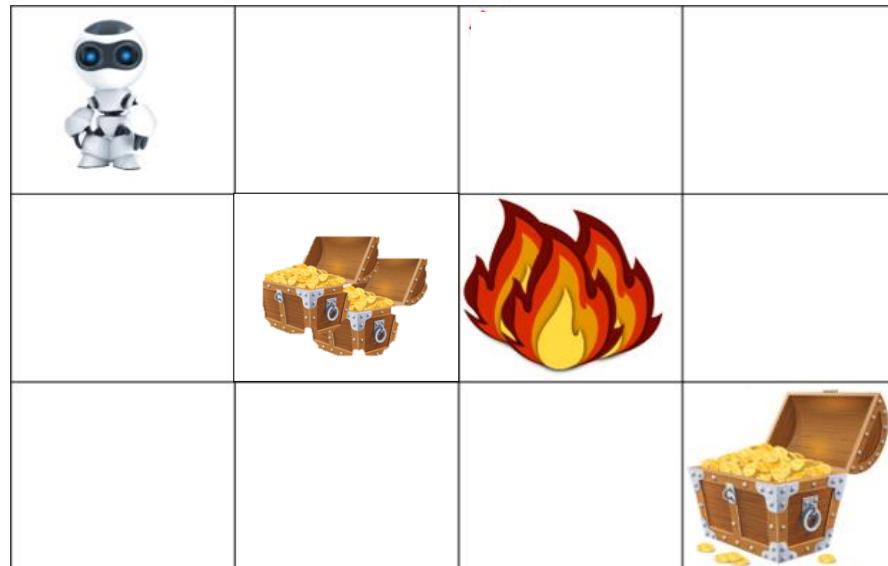
	Kiri	Kanan	Atas	Bawah
(1,1)	Q-Value	Q-Value	Q-Value	Q-Value
(1,2)	Q-Value	Q-Value	Q-Value	Q-Value
(1,3)	Q-Value	Q-Value	Q-Value	Q-Value
(1,4)	Q-Value	Q-Value	Q-Value	Q-Value
(2,1)	Q-Value	Q-Value	Q-Value	Q-Value
(2,2)	Q-Value	Q-Value	Q-Value	Q-Value
(2,3)	Q-Value	Q-Value	Q-Value	Q-Value
(2,4)	Q-Value	Q-Value	Q-Value	Q-Value
(3,1)	Q-Value	Q-Value	Q-Value	Q-Value
(3,2)	Q-Value	Q-Value	Q-Value	Q-Value
(3,3)	Q-Value	Q-Value	Q-Value	Q-Value
(3,4)	Q-Value	Q-Value	Q-Value	Q-Value

How Q Learning work?

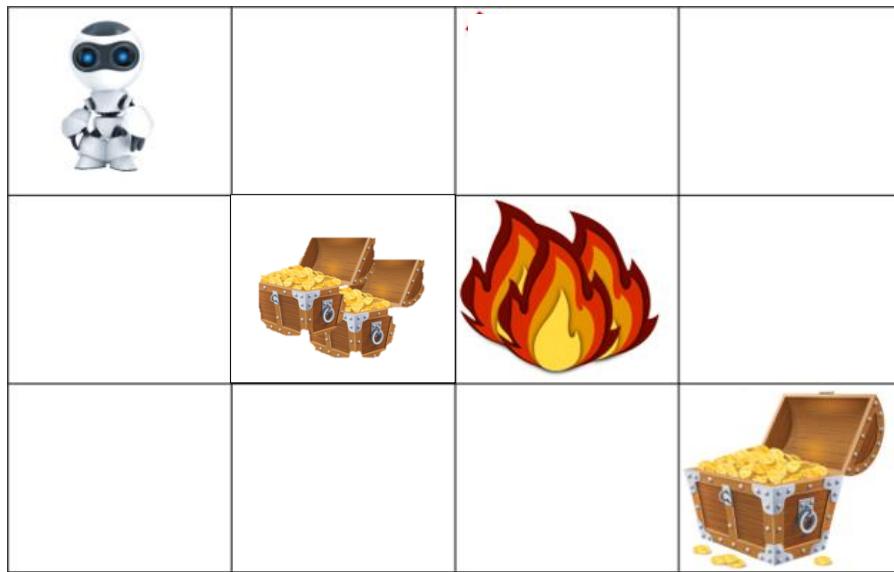


Studi Kasus

Studi kasus kali ini kita akan membantu robot Orbit untuk menemukan harta karun yang paling banyak dengan step yg sedikit dan menghindari supaya tidak terkena api azab.



1. Definisi Reward dan Q Table



Reward Table :

State	Reward
Empty	-1
Fire	-10
1 Harta Karun	+10
2 Harka Karun	+25

1. Definisi Reward dan Q Table (cont)



Actions

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

States

2. Choose Action

Pada dasarnya dalam algoritma Q-Learning, *agent* akan memilih *action* berdasarkan Q-Table. *Agent* akan memilih *action* yang mempunyai Q-Value paling besar berdasarkan *state* saat ini. Tapi.... Hal tersebut akan terdengar aneh untuk Langkah pertama, bukan? Karena pada Langkah pertama semua Q-Value yg ada pada Q-Table bernilai 0, jadi gimana cara *agent* memilih *action*?

Nah, untuk jawab pertanyaan ini kita akan coba mengingat Kembali tentang Eksplorasi dan Eksplotasi.

2. Choose Action : Eksplorasi vs Eksplotasi

Eksplorasi adalah suatu metode pemilihan *action* dimana *agent* akan melakukan pemilihan *action* secara *random* dengan tujuan ia bisa mengetahui informasi tentang *environment* secara mendalam.

Sedangkan **Eksplotasi** adalah sebuah metode pemilihan *action* dengan memilih *action* yang mempunyai *return* (dalam hal ini Q-Value) paling besar.

Lalu, pasti teman teman semua berfikir kalau pemilihan *action* dengan metode eksplotasi adalah yg terbaik. Tentu saja tidak begituu! Loh kok gituu?? Kenapa???

2. Choose Action : Eksplorasi vs Eksplotasi (cont)

Mari kita bayangkan *environtment* kita adalah studi kasus kita pada kali ini.

Bagaimana kalau *agent* (robot orbit) menemukan terlebih dahulu satu harta karun, lalu kita menggunakan metode eksplotasi dalam memilih *action*? Maka yang terjadi adalah *agent* akan cenderung memilih *action* yg akan menuntun dia ke satu harta karun dibandingkan dua harta karun, bam!!

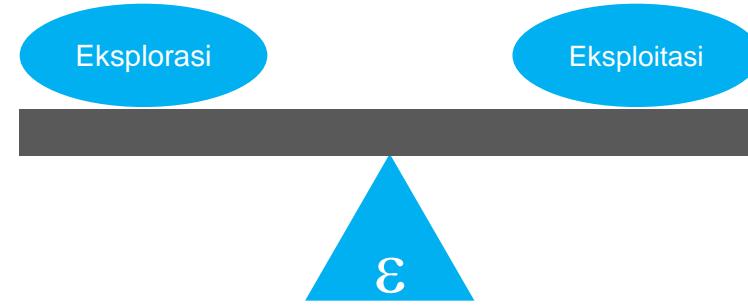


Lalu apakah metode eksplorasi adalah yg terbaik? Oh tentu tidak juga! Jika kita menggunakan metode eksplorasi (memilih *action* secara random) secara terus menerus maka kita akan kesulitan untuk mencapai solusi yang optimum.

Lalu solusinya bagaimana? Jawabannya adalah metode **epsilon-greedy exploration**

2. Choose Action : epsilon-greedy exploration

epsilon-greedy exploration digunakan untuk menyeimbangkan antara eksplorasi dan eksplotasi menggunakan nilai epsilon (ε). ε biasa disebut dengan rate eksplorasi.

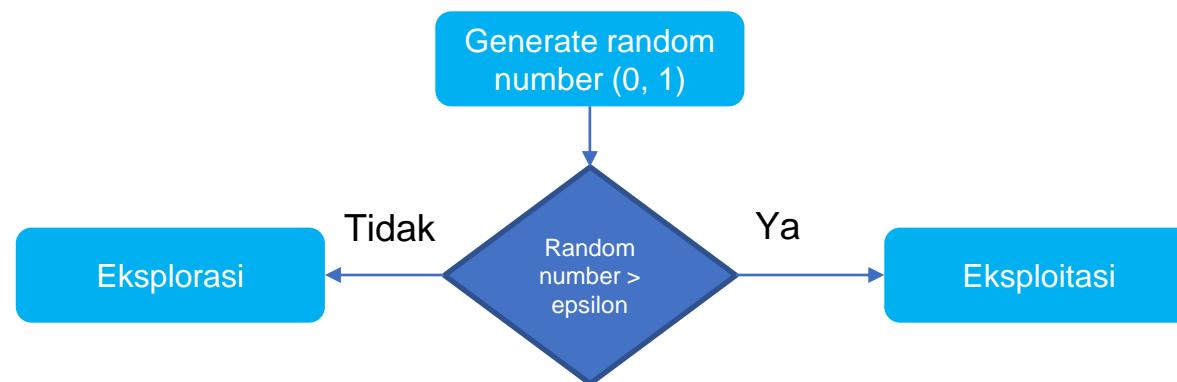


Pertama kita set nilai $\varepsilon = 1$, itu artinya pada episode – episode awal kita akan menggunakan metode eksplorasi dalam memilih *action* dengan tujuan agar *agent* dapat mengenali *environtment*-nya dengan baik.

2. Choose Action : epsilon-greedy exploration (cont)

Lalu dengan seiring berjalannya waktu nilai ϵ akan kita kurangi dengan rate tertentu, sehingga *agent* akan cenderung memilih *action* menggunakan metode Eksplorasi.

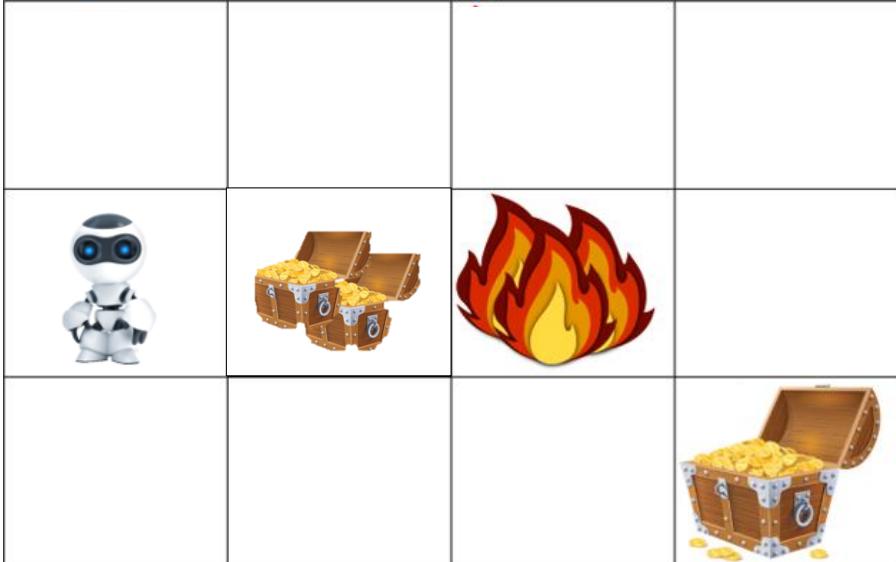
Lalu, untuk memberi tahu *agent* kapan harus pakai eksplorasi dan kapan harus pakai eksplotasi, kita akan generate nomor acak dari 0 sampai 1. Jadi jika digambarkan dengan flowchart maka akan menjadi seperti ini.



2. Choose Action

Kembali ke Langkah kedua yaitu pemilihan *action*, maka sesuai dengan metode **epsilon-greedy exploration**, kita akan set nilai $\epsilon = 1$ dan untuk awal kita akan memilih *action* secara random. Kita misalkan *agent* memilih *action* ke bawah.

3. Perform Action and Get Reward



Reward = -1

4. Update Q-Table

Setelah eksekusi *action* dan mendapatkan *reward*, saatnya kita mengupdate Q-Table dengan mengganti Q-Value yg lama dengan Q-Value yang baru, untuk mengupdate Q-Table dan menghitung Q-Value yg baru kita dapat menggunakan formula berikut :

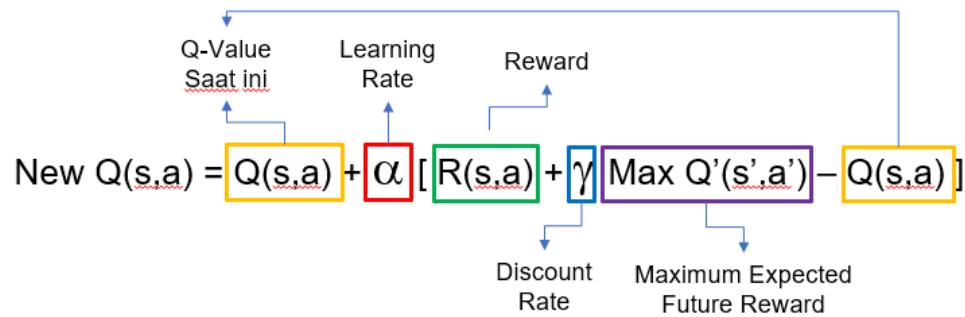
$$\text{New } Q(s,a) = Q(s,a) + \alpha [R(s,a) + \gamma \text{Max } Q'(s',a') - Q(s,a)]$$

The diagram illustrates the Q-learning update equation with colored boxes and arrows:

- Q-Value Saat ini**: Points to the first $Q(s,a)$ term.
- Learning Rate**: Points to the α term.
- Reward**: Points to the $R(s,a)$ term.
- Discount Rate**: Points to the γ term.
- Maximum Expected Future Reward**: Points to the $\text{Max } Q'(s',a')$ term.

4. Update Q-Table (cont)

Kita akan coba update Q-Table untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$:



$$\text{New } Q(\text{s},\text{a}) = Q(\text{s},\text{a}) + \alpha [R(\text{s},\text{a}) + \gamma \text{Max } Q'(\text{s}',\text{a}') - Q(\text{s},\text{a})]$$

$$\text{Max } Q'(\text{s}',\text{a}') = \text{Max}(q((2,1), \text{kiri}), q((2,1), \text{kanan}), q((2,1), \text{atas}), q((2,1), \text{bawah}))$$

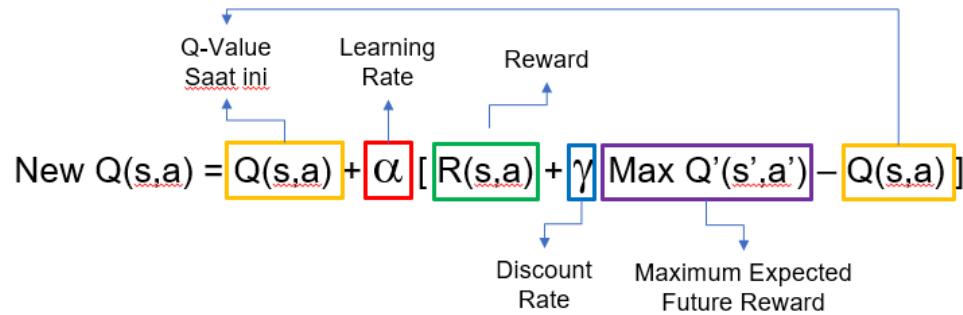
$$\text{Max } Q'(\text{s}',\text{a}') = \text{Max}(0, 0, 0, 0)$$

$$\text{Max } Q'(\text{s}',\text{a}') = 0$$

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

4. Update Q-Table (cont)

Kita akan coba update Q-Table untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$:



$$\max Q'(s',a') = 0$$

$$\text{New } Q(s,a) = Q(s,a) + \alpha [R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$$

$$\text{New } Q((1,1), \text{bawah}) = 0 + 0.7 [-1 + 0.99 (0) - 0]$$

$$\text{New } Q((1,1), \text{bawah}) = 0 + 0.7 [-1]$$

$$\text{New } Q((1,1), \text{bawah}) = -0.7$$

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

4. Update Q-Table (cont)

Kita akan coba update Q-Table untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$:

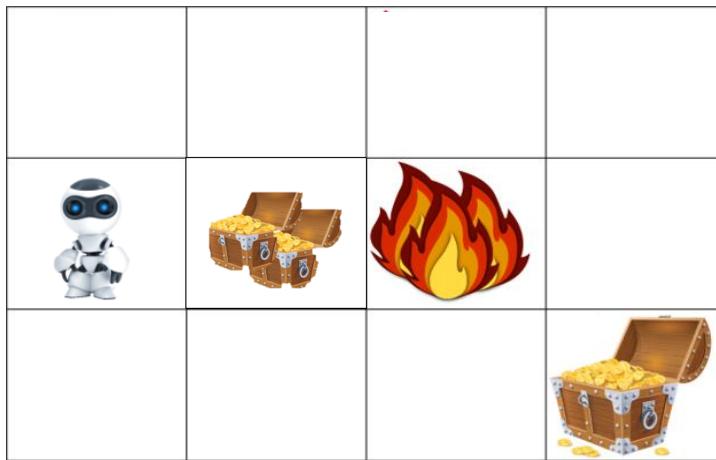
$$\text{Max } Q'(s',a') = 0$$

$$\text{New } Q(s,a) = Q(s,a) + \alpha [R(s,a) + \gamma \text{ Max } Q'(s',a') - Q(s,a)]$$

$$\text{New } Q((1,1), \text{ bawah}) = 0 + 0.7 [-1 + 0.99 (0) - 0]$$

$$\text{New } Q((1,1), \text{ bawah}) = 0 + 0.7 [-1]$$

$$\text{New } Q((1,1), \text{ bawah}) = -0.7$$



	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	-0.7
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

Jadi, kita sudah melakukan kalkulasi untuk Q-Value yg baru untuk *state* (1,1) dan *action* ke bawah, dan kita juga udah store value tersebut ke Q-Tabel.

Kita sudah selesai melakukan proses Q-Learning untuk satu step. Proses ini (Langkah 2 – 4) akan dilakukan sampai beberapa episode. Sampai Q-Tabel yang kita punya konvergen, sehingga kita menemukan Policy yang paling optimal

Limitation of Q Learning

Pada game robot karun, *environment* kita sangat *simple* sekali dengan hanya memiliki 12 *states* dan 4 *actions*. Artinya kita hanya akan punya Q-Table sebesar 12x4 atau ada 48 Q-Values yang harus diupdate dalam setiap iterasi.

Bisa kita bayangkan kalau kita punya *envinronment* dengan ukuran *states* dan *action* yang besar? Seperti Dota 2 contohnya. *States* dan *action* yang ada pada game tersebut pasti akan besar sekali, hal tersebut akan membuat ukuran Q-Table kita akan menjadi besarr sekalii.

Lalu bagaimana kalau kita ingin bereksperimen dengan *envinronment* yang lebih besar dari game karun? Jawabannya adalahh....



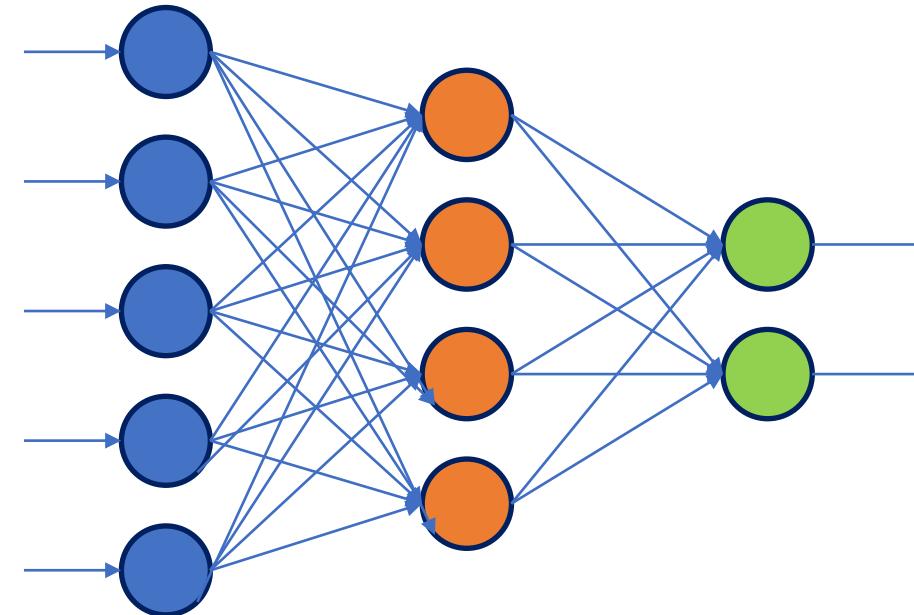
02

Deep Q Learning

- Deep Q Learning
- Deep Q Network
- Experience Replay and Replay Memory
- How we Train DQN and Calculate Loss in DQN
- How Train DQN

Deep Q Learning

Pada *Deep Q Learning* kita akan menggantikan Q-Table menggunakan sebuah *Neural Network* yang biasa disebut dengan *Deep Q Network* atau DQN.



Deep Q Network

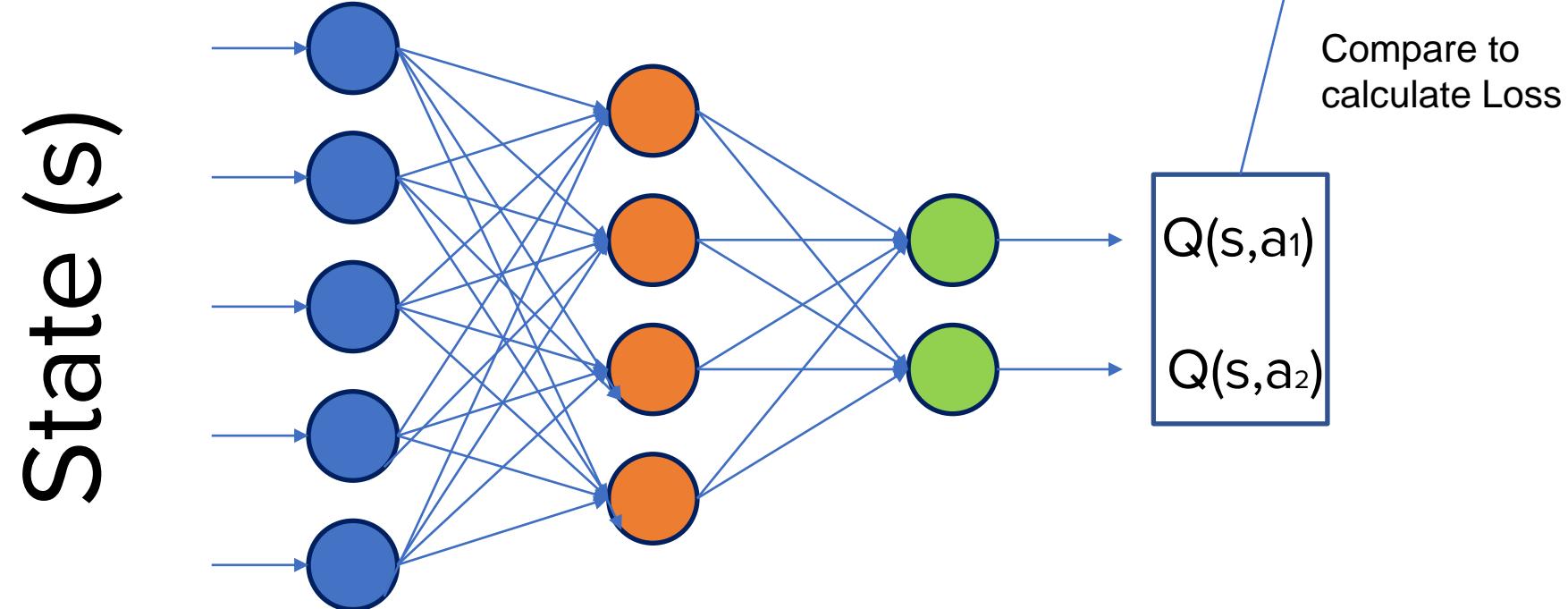
Deep Q Network adalah sebuah NN yang menerima *states* yg diberikan oleh *environment* sebagai input, lalu DQN akan menghasilkan output estimasi Q Values pada setiap *actions* yang dapat diambil pada *state* tersebut. Tujuan dari NN ini adalah untuk menghasilkan aproksimasi Q Function yg optimal. Seperti yg kita tau pada slide sebelumnya kalau Optimal Q Function adalah seperti ini :

$$q_* (s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_* (s', a') \right]$$

Loss pada NN ini adalah dengan membandingkan antara Q-Values dari output dengan target Q-Values yang didapatkan dari persamaan diatas

Deep Q Network (cont)

$$q_*(s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') \right]$$



Goal dari NN ini adalah untuk minimize loss, lalu setelah menghitung loss bobot pada *network* akan diupdate menggunakan *stochastic gradient descent* dan *backpropagation* seperti *neural network* pada umumnya.

Experience Replay and Replay Memory

Dalam proses *training* DQN kita akan menggunakan sebuah teknik yg dinamakan dengan ***experience replay***. Dengan *experience replay*, kita menyimpan *experience* dari *agent* untuk setiap *time step* ke dalam sebuah wadah yang bernama ***replay memory***.

Kita dapat merepresentasikan *experience* dari *agent* untuk setiap t sebagai e_t . Pada time t, *agent experience* e_t didefinisikan sebagai tuple berikut :

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

Tuple ini menyimpang *state* (s_t), *action* (a_t) yang diambil, reward (r_{t+1}) yang didapatkan oleh *agent* dan *state* selanjutnya (s_{t+1}).

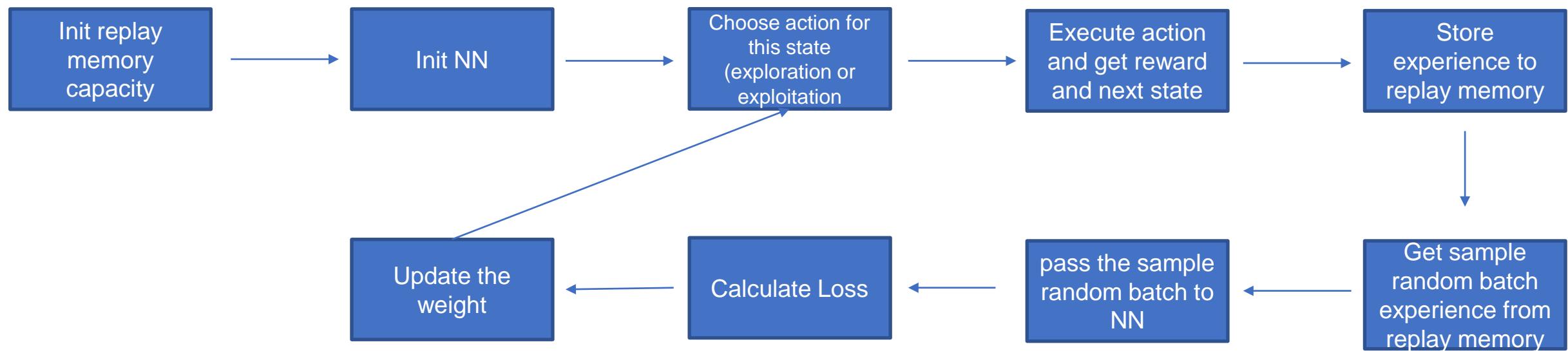
Experience Replay and Replay Memory (cont)



Secara teori seluruh *experience agent* pada setiap *time step* akan disimpan pada *replay memory*. Sebenarnya dalam praktiknya, kita akan mendefinisikan besaran *experience* yang dapat ditampung oleh *replay memory* sebesar N, dan kita hanya akan menyimpan N terakhir *experience* dari *agent*.

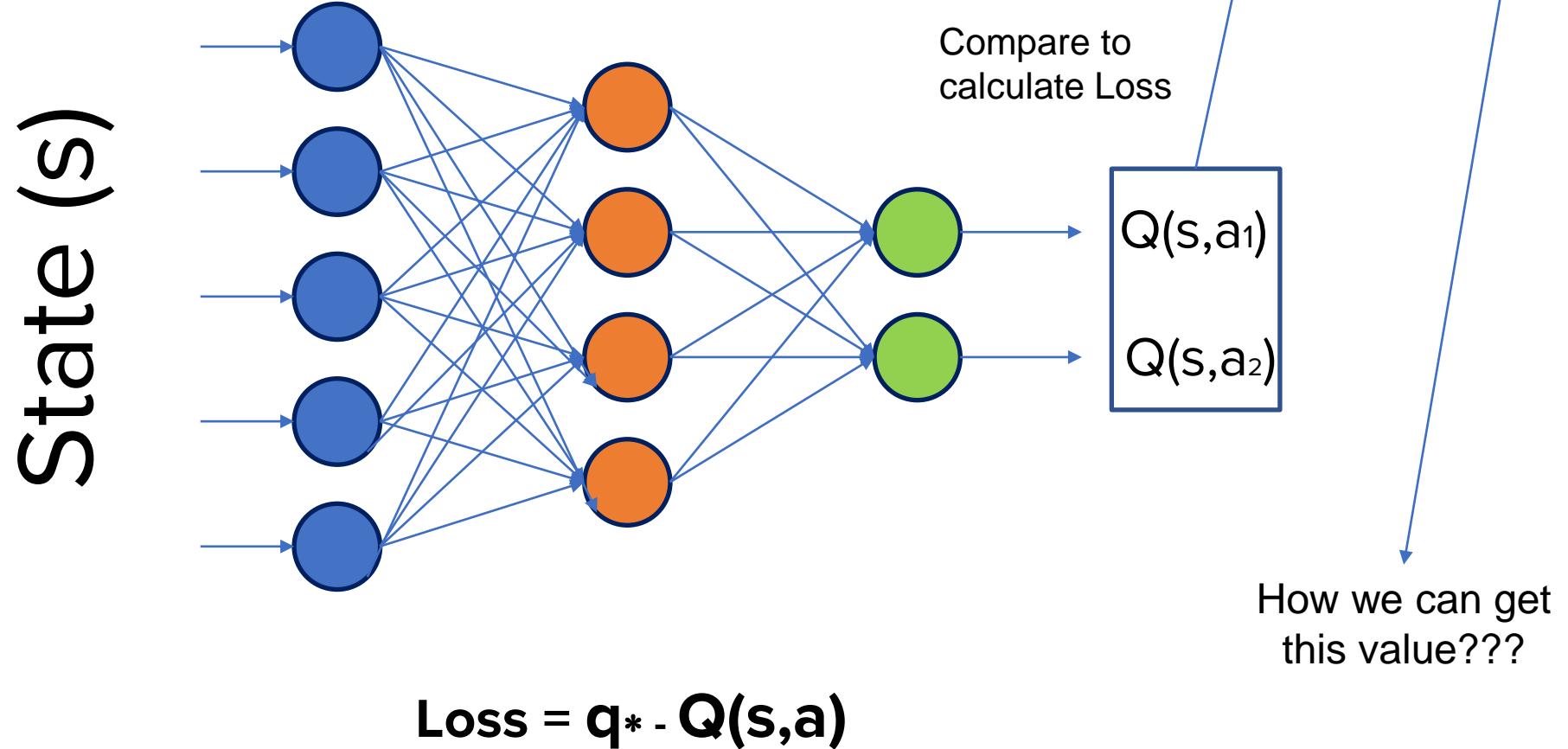
Source gambar : deeplizard

How we Train the DQN?



How We Calculate Loss on DQN?

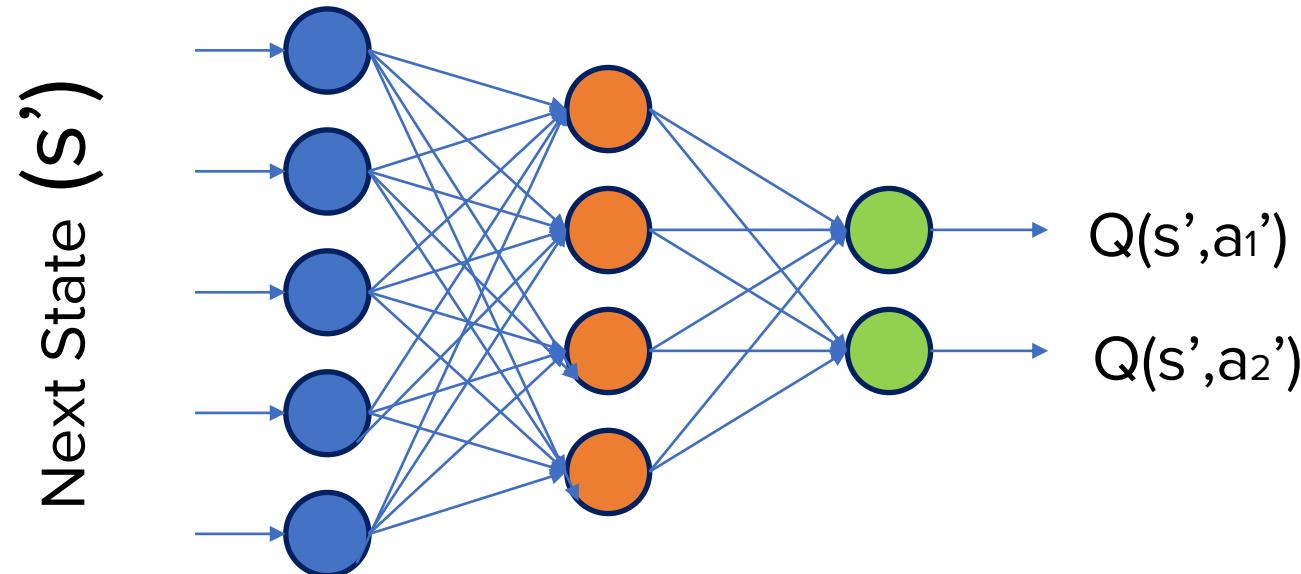
$$q_*(s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') \right]$$



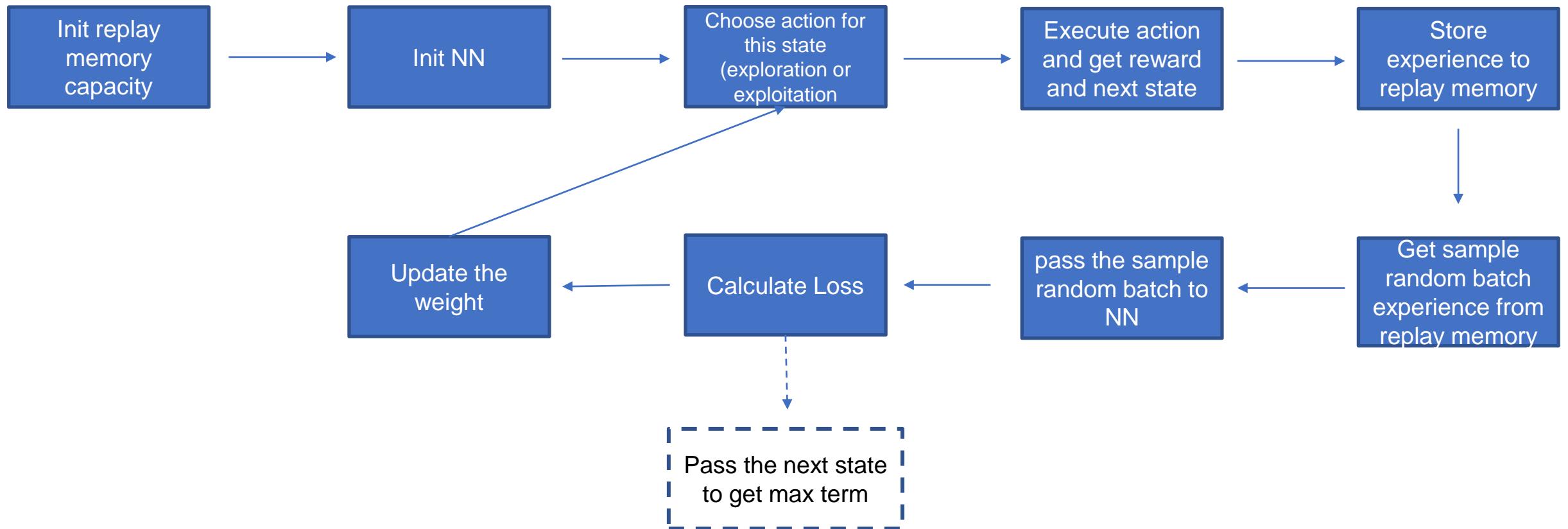
How We Calculate Loss on DQN? : Calculate the Max Term

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

Pada Q Learning kita dapat mendapatkan value diatas dengan melihatnya di Q-Table, tapi itu adalah cara yg kunooo! Pada Deep Q Learning kita akan mendapatkan value tersebut dengan bantuan DQN kita



How to train DQN



Algoritma DQN

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Pseudocode DQN

```
Initialize  $Q$  function with random weight  $\theta$ 
Initialize  $\hat{Q}$  function with random weight  $\theta' = \theta$ 
Initialize empty replay memory  $D$ 

for  $episode = 1..M$  do
    Initialize environment  $s \leftarrow env.reset()$ 
    for  $t = 1..T$  do
        > Collect observation from the env:
         $a \leftarrow \epsilon\text{greedy}(\phi(s))$ 
         $s', r, d \leftarrow env(a)$ 
        > Store the transition in the replay buffer:
         $\varphi \leftarrow \phi(s)$ ,  $\varphi' \leftarrow \phi(s')$ 
         $D \leftarrow D \cup (\varphi, a, r, \varphi', d)$ 
        > Update the model using (5.4):
        Sample a random minibatch  $(\varphi_j, a_j, r_j, \varphi'_j, d_j)$  from  $D$ 
         $y_j = \begin{cases} r_j & \text{if } d_{j+1} = \text{True} \\ r_j + \gamma \max_{a'} \hat{Q}_{\theta'}(\phi_{j+1}, a') & \text{otherwise} \end{cases}$ 
        Perform a step of GD on  $(y_j - Q_{\theta}(\varphi_j, a_j))^2$  on  $\theta$ 
        > Update target network:
        Every  $C$  steps  $\theta' \leftarrow \theta$  (i.e.  $\hat{Q} \leftarrow Q$ )
         $s \leftarrow s'$ 
    end for

end for
```

Rangkuman

1. Q-learning termasuk dalam kategori Model-free RL algorithm. Menggunakan Bellman Equation untuk perhitungan dan bersifat online action-value function learning dengan exploration policy. Sehingga kita perlu belajar konsep dasar RL sebelum mempelajari Q-learning.
2. Pada Q-learning, agent belajar menggunakan evaluation function yang bergantung pada sekumpulan state dan sekumpulan action.
3. Pada Q-learning dilakukan iterasi:
 - Policy iteration
 - Value iteration
4. Di tahap awal, elemen matrix Q-table akan ditentukan bernilai 0 semuanya. Q-table akan di update sejalan dengan bertambahnya jumlah episode.
5. Secara prinsip semakin banyak iterasi episode akan semakin baik Q-table-nya.
6. Q-table ini dapat dianalogikan seperti 'memori' bagi agent. Agent dapat menentukan path optimal berdasarkan nilai yang tertera pada Q-table.

Code



<https://github.com/alien087/robot-karun>

This presentation is made by



Wayan Dadang S.T.

Wijaya Yudha Atmaja

Dino Febriyanto S.Kom.

Ryan Satria Wijaya S.T.

Oetomo



References

- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2020
- Sudharsan Ravichandiran. Hands-On Reinforcement Learning with Python. 2018
- Andrea Lonza. Reinforcement Learning Algorithms with Python. 2019
- Playing Atari with Deep Reinforcement Learning by Deep Mind Technologies (paper)(<https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf>)
- Jie-Han Chen. Temporal-difference Learning. National Cheng Kung University, Taiwan: Slide PPT
- Human-level control through deep reinforcement learning (paper)
(<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>)



TERIMA KASIH

Orbit Future Academy

PT Orbit Ventura Indonesia
Center of Excellence (Jakarta Selatan)
Gedung Veteran RI, Lt.15
Unit Z15-002, Plaza Semanggi
Jl. Jenderal Sudirman Kav.50, Jakarta
12930, Indonesia

- Jakarta Selatan/Pusat
- Jakarta Barat/BSD
- Kota Bandung
- Kab. Bandung
- Jawa Barat

Hubungi Kami

Director of Sales & Partnership
ira@orbitventura.com
+62 858-9187-7388

Social Media

-  Orbit Future Academy
-  @OrbitFutureAcademyIn1
-  OrbitFutureAcademy
-  Orbit Future Academy

Welcome Everyone

AI Mastery Course



Modul Domain AI

Reinforcement Learning

Bagian

Robotics





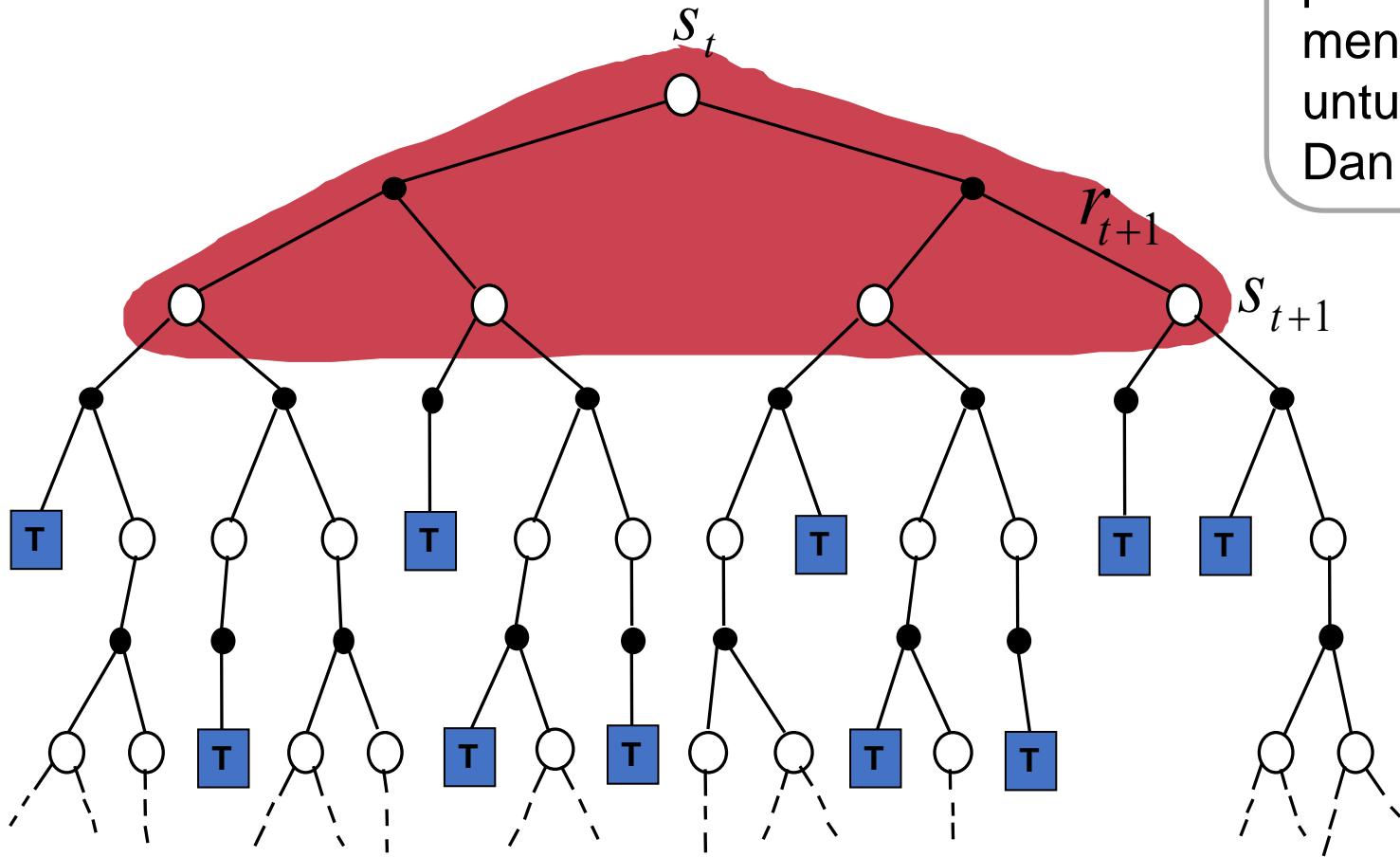
Learning Objectives

- Mengetahui Dasar dari Robotika
- Mengetahui Kegunaan Konsep Machine Learning untuk Robotika
- Mengimplementasikan Reinforcement Learning pada Robotika



Dynamic Programming (Recap)

$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$

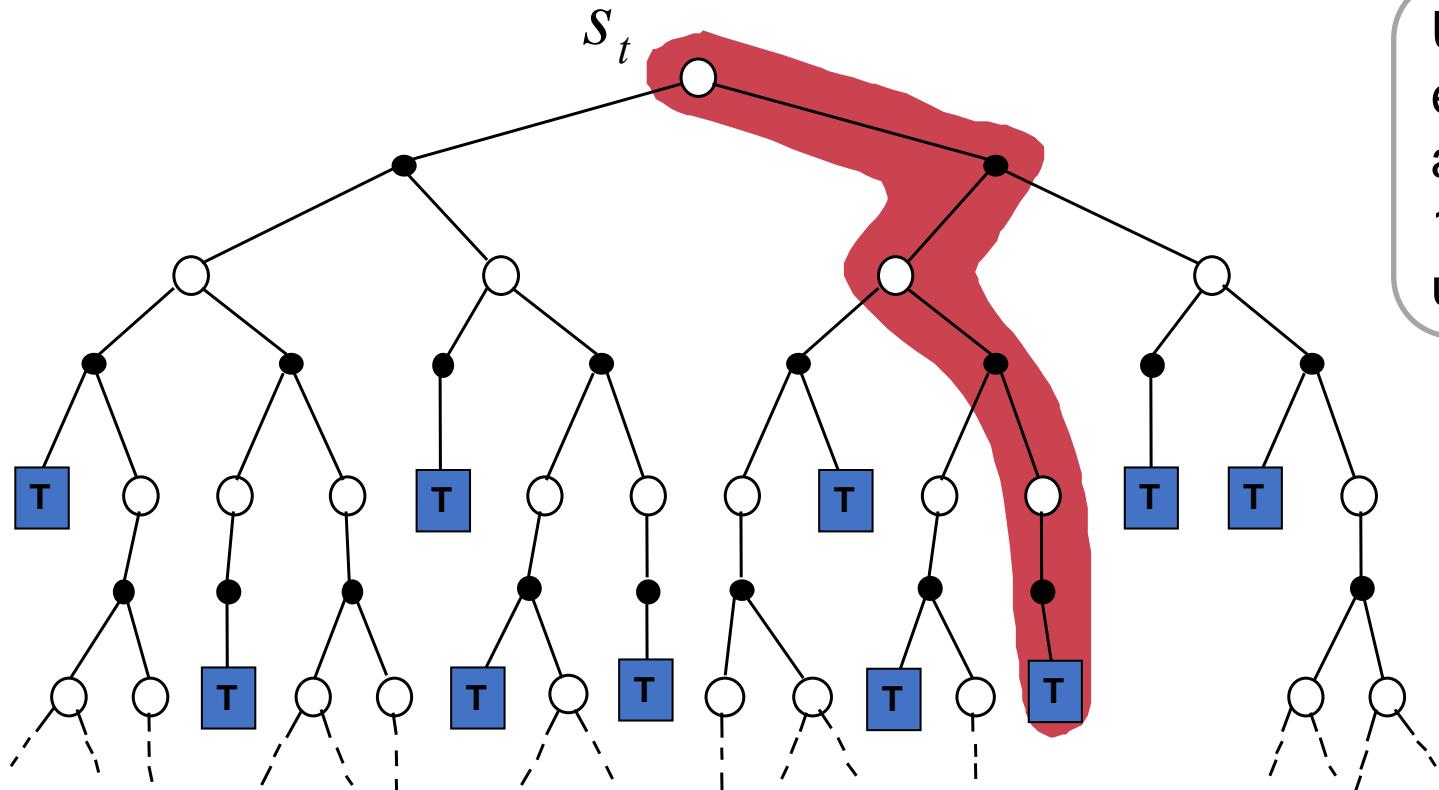


Update value state $V(S_t)$ per step. Jadi pada DP algoritma **tidak harus** menyelesaikan 1 episode secara utuh untuk meng-update value state $V(s)$. Dan bersifat bootstrapping.

Simple Monte Carlo (Recap)

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Dimana G_t adalah *actual return* terhadap state S_t

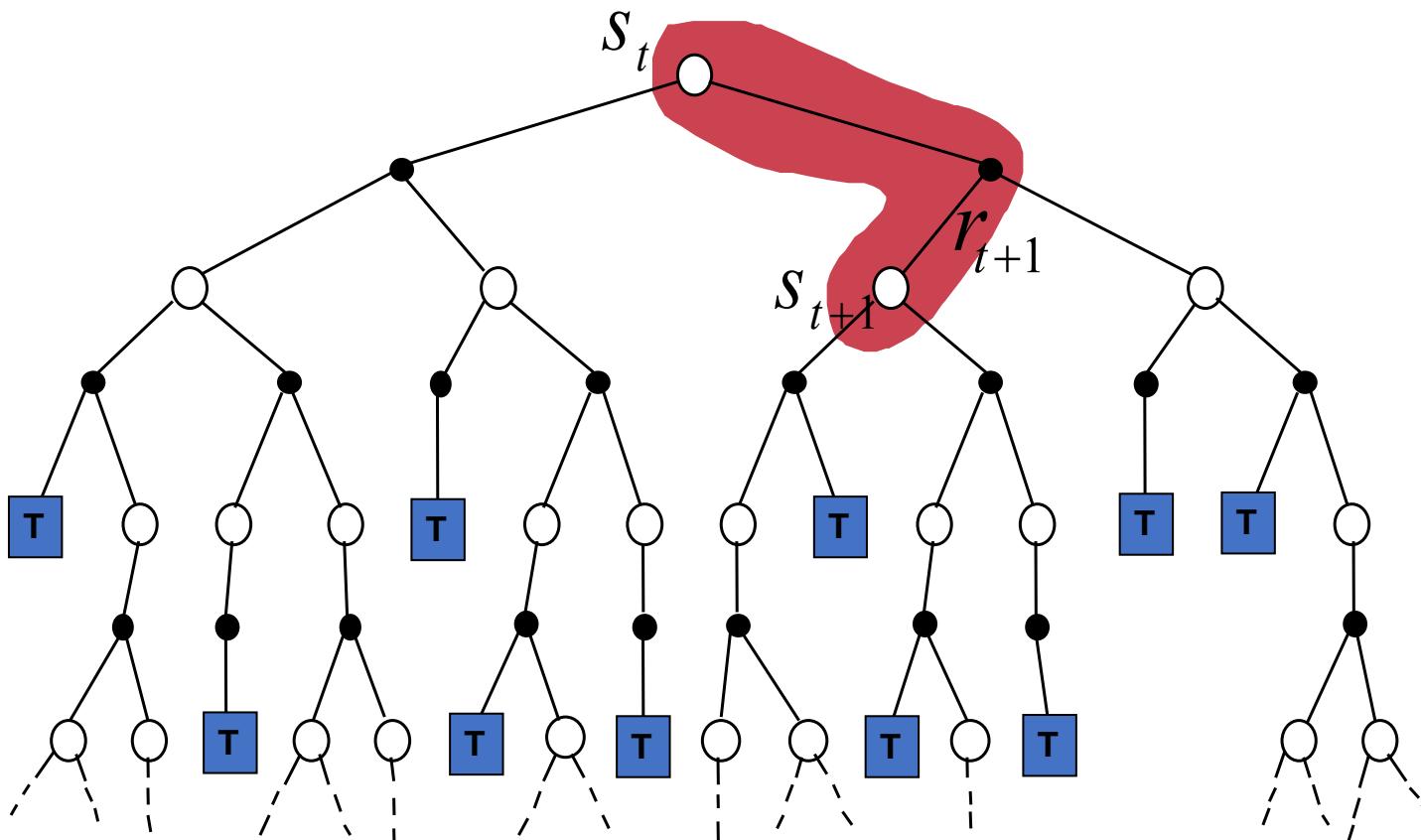


Update value state $V(S_t)$ setelah episode selesai. Jadi pada MC algoritma **harus** menyelesaikan dahulu 1 episode secara utuh untuk meng-update value state $V(s)$.

Simplest TD Method (Recap)

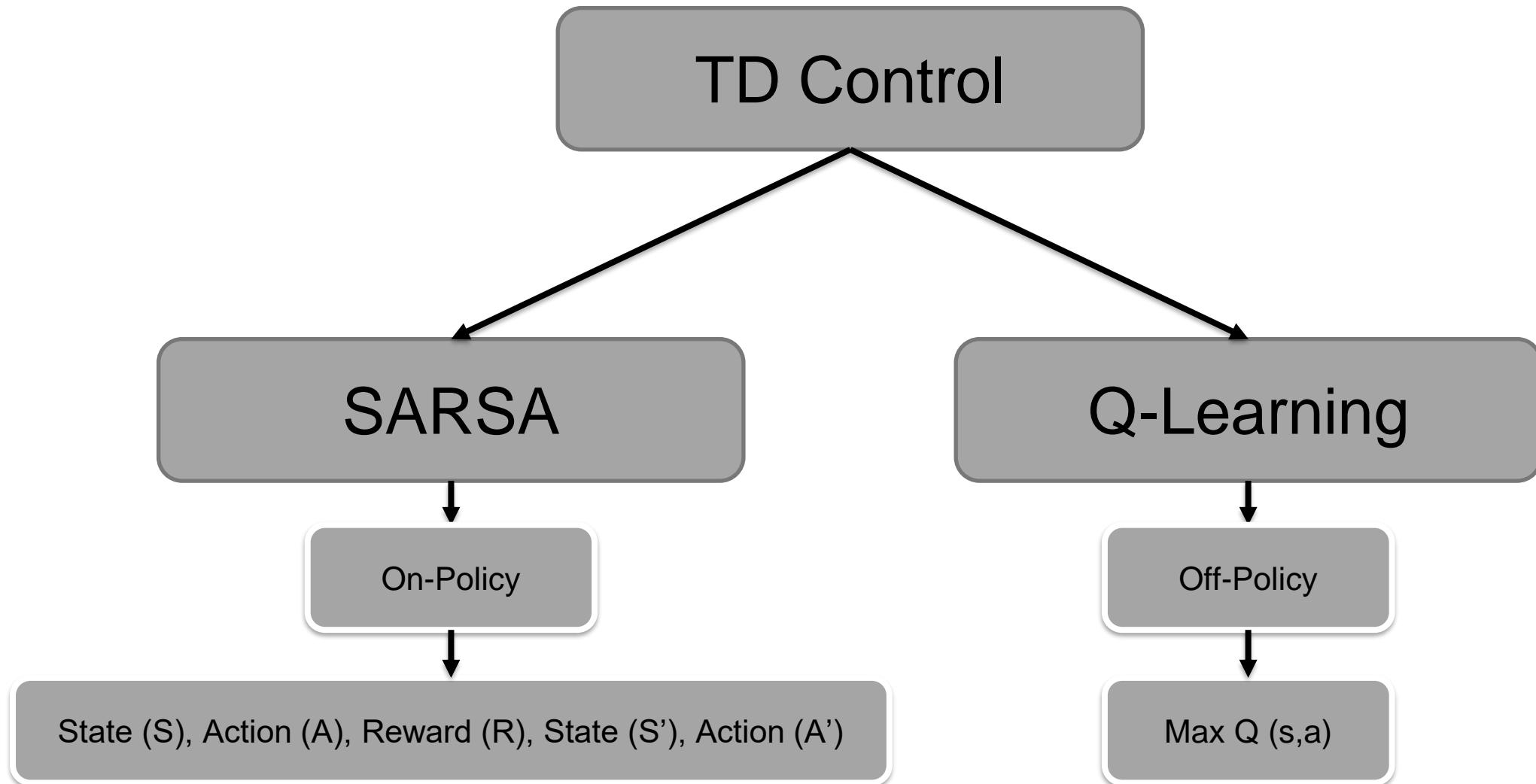
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

TD target: an estimate of the return

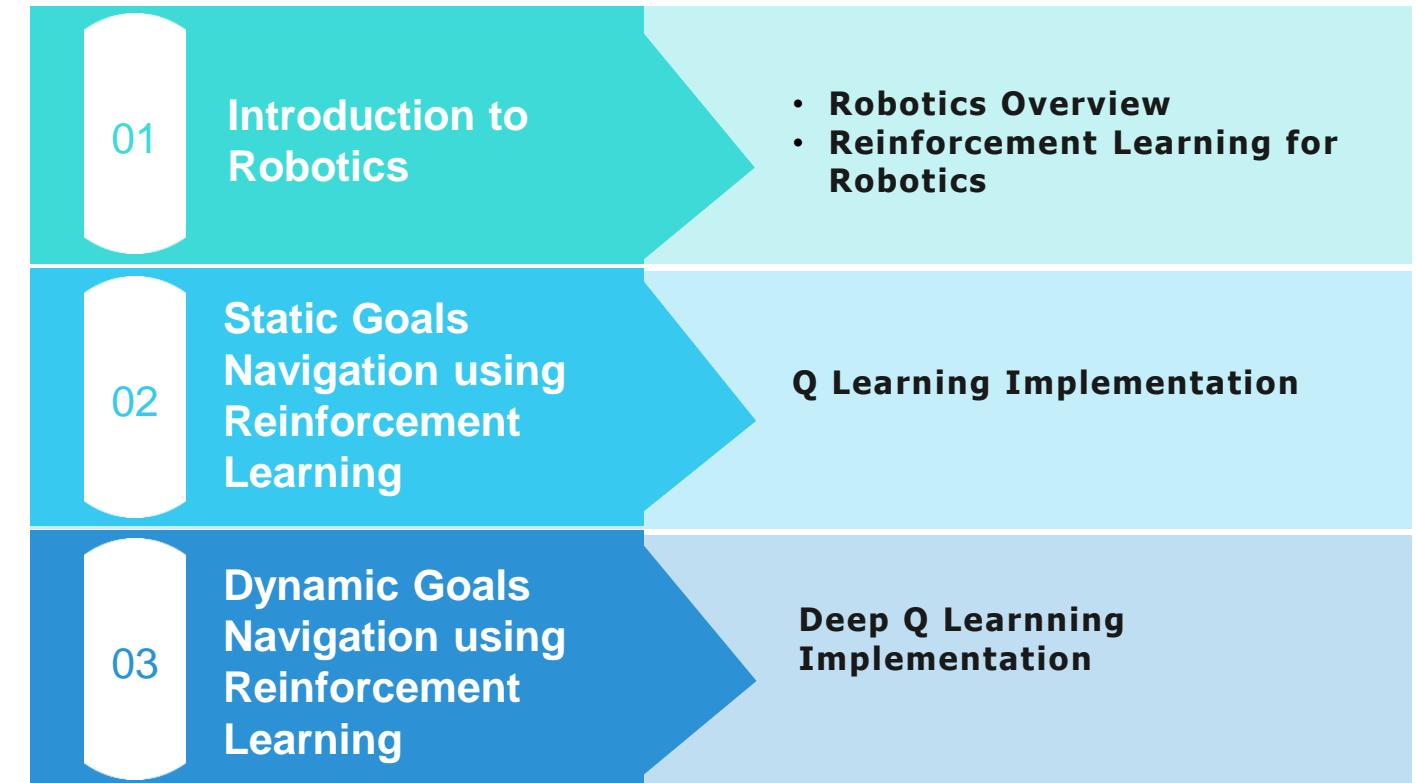


Update value state $V(S_t)$ per step seperti dynamic programming. Jadi pada TD algoritma **tidak harus** menyelesaikan 1 episode secara utuh untuk meng-update value state $V(s)$.

Temporal Difference Learning (Recap)



Agenda





01

Introduction to Robotics

- **Robotics Overview**
- **Reinforcement Learning for Robotics**

Robotics Overview

What is Robotics ?

- Robotics adalah suatu disiplin ilmu yang mempelajari tentang konsep suatu robot

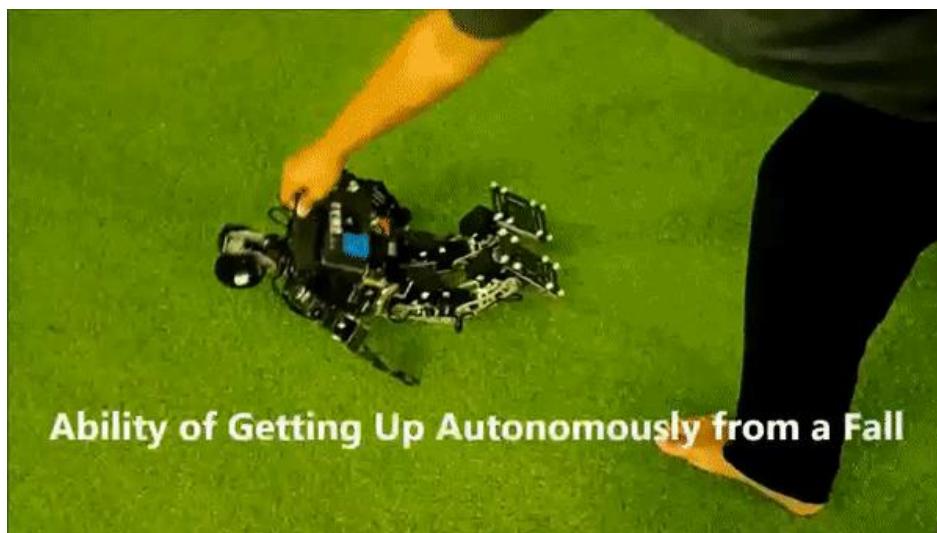
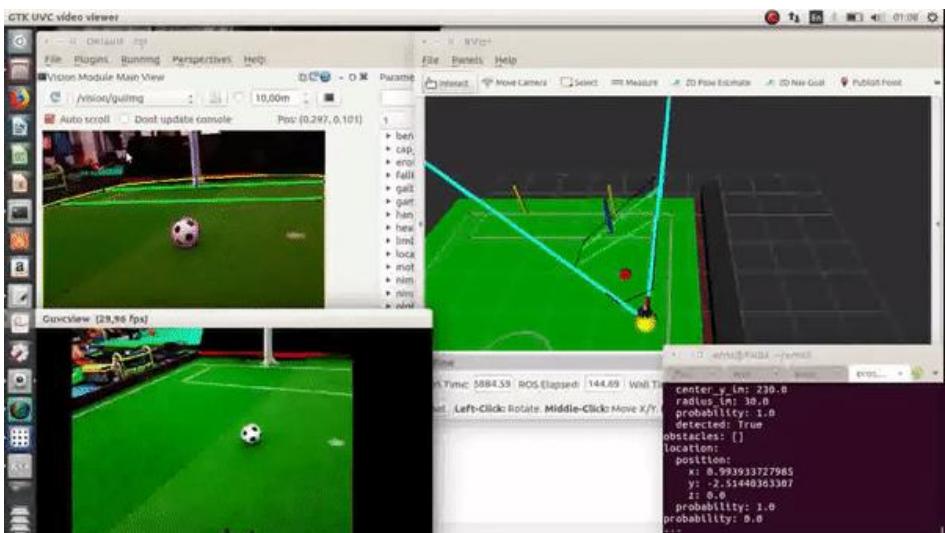
What is Robot ?

- Robot merupakan mesin yang beroperasikan secara otomatis yang menggantikan usaha manusia, meskipun mungkin tidak menyerupai manusia dalam penampilan atau melakukan fungsi dengan cara yang mirip manusia.



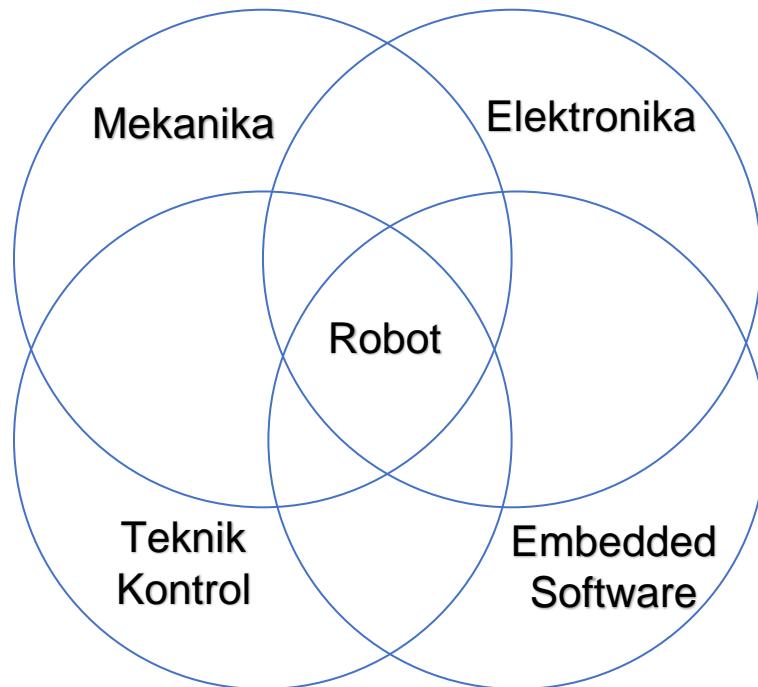
Robotics Overview

Contoh beberapa aplikasi Robot:



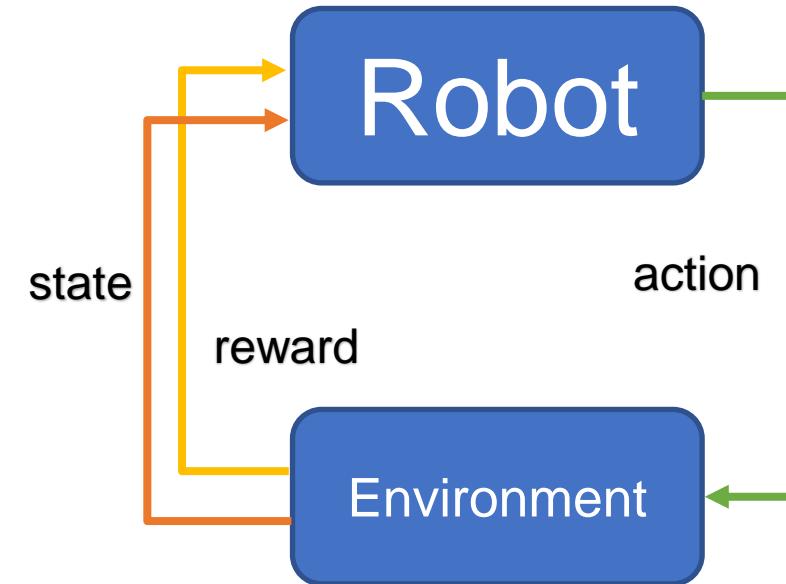
Robotics Overview

Interdisiplin Robotika :

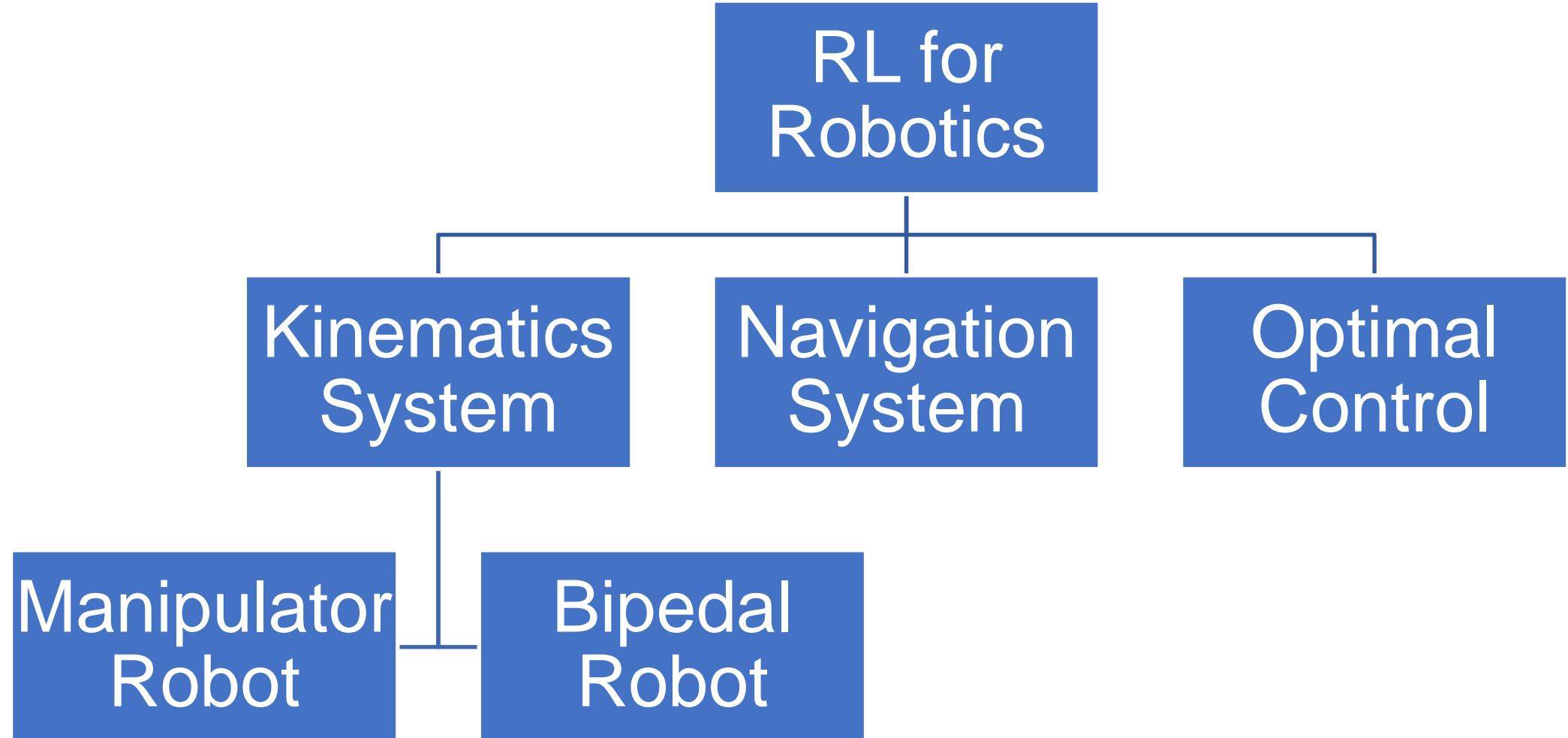


Reinforcement Learning for Robotics

- Apakah RL bisa diterapkan ke dalam Robotics?
- Kira kira apa saja penerapan RL ke dalam Robotics?



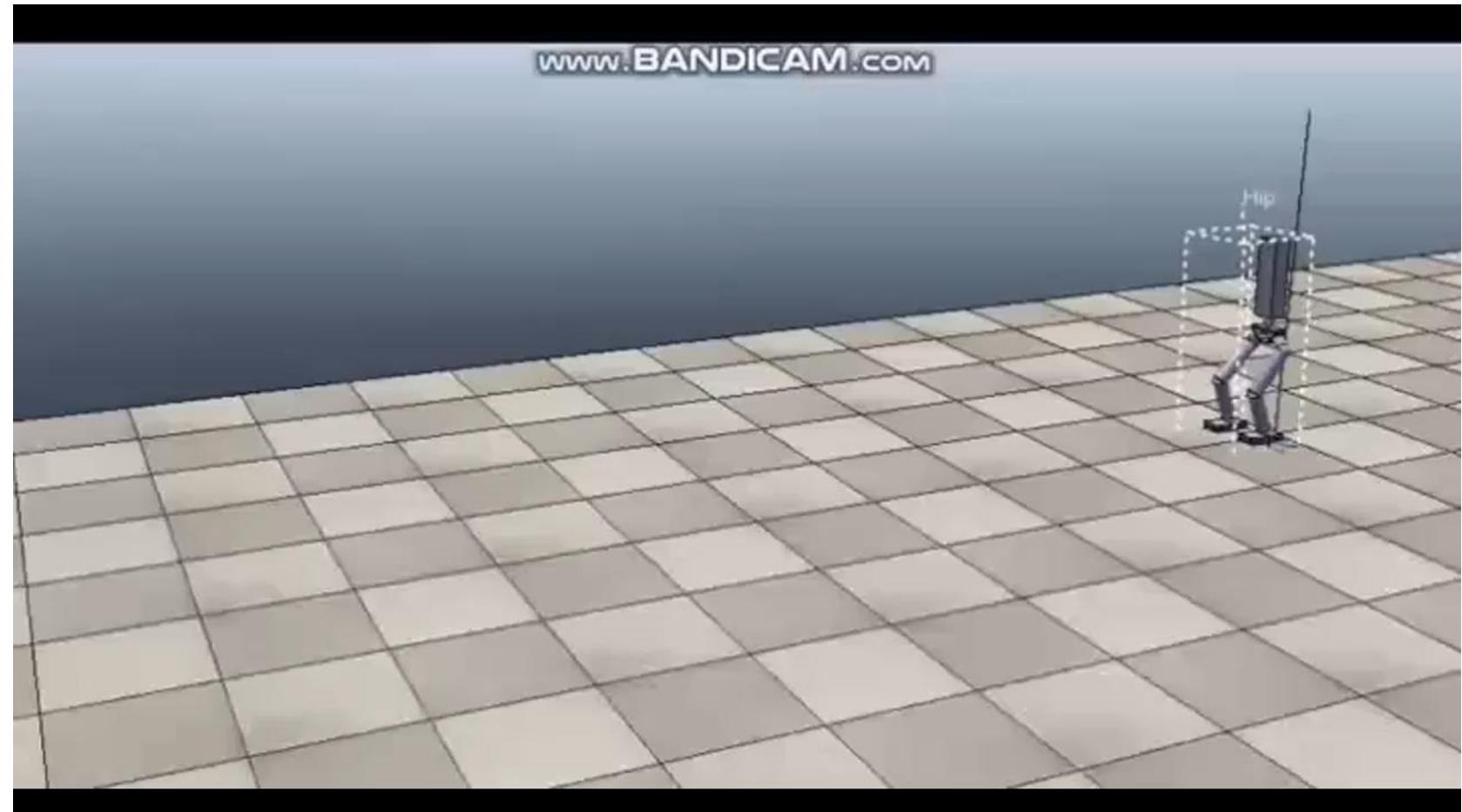
Reinforcement Learning for Robotics



Reinforcement Learning for Robotics

➤ Kinematics System

- State : Total Joints or Degrees of Freedom of the robot on radian.
- Action: X,Y,Z Cartesian Position of each legs (6 actions)



Reinforcement Learning for Robotics

➤ Navigation System

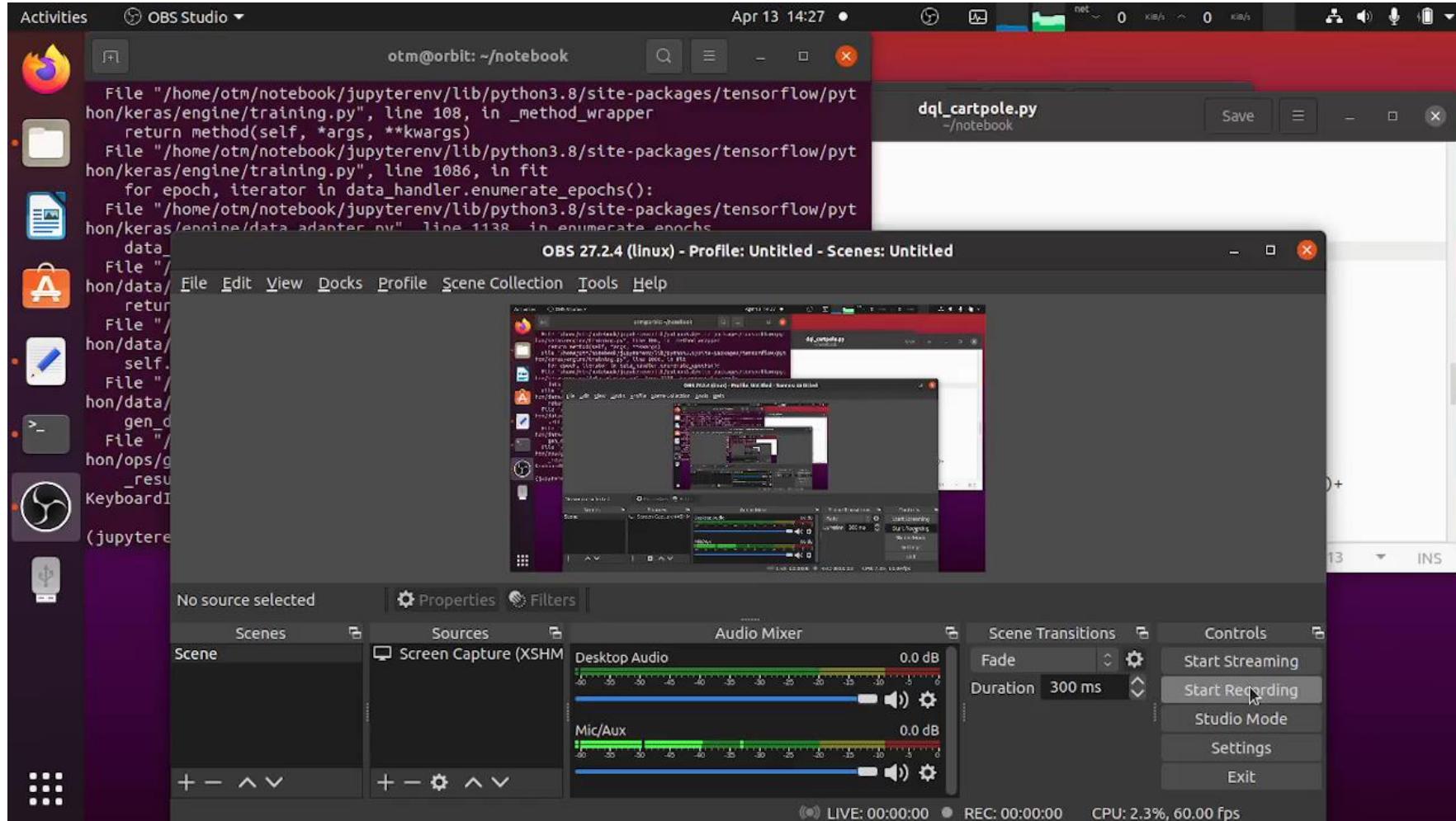
- Motion planning, trajectory optimization, dynamic pathing, controller optimization, and scenario-based learning policies are the application of self driving car using reinforcement learning.
- Places, Driveable zones, Obstacle avoidance, Velocity, and Direction can be a varible for defining the state and action on reinforcement learning model for self driving car.

Source:

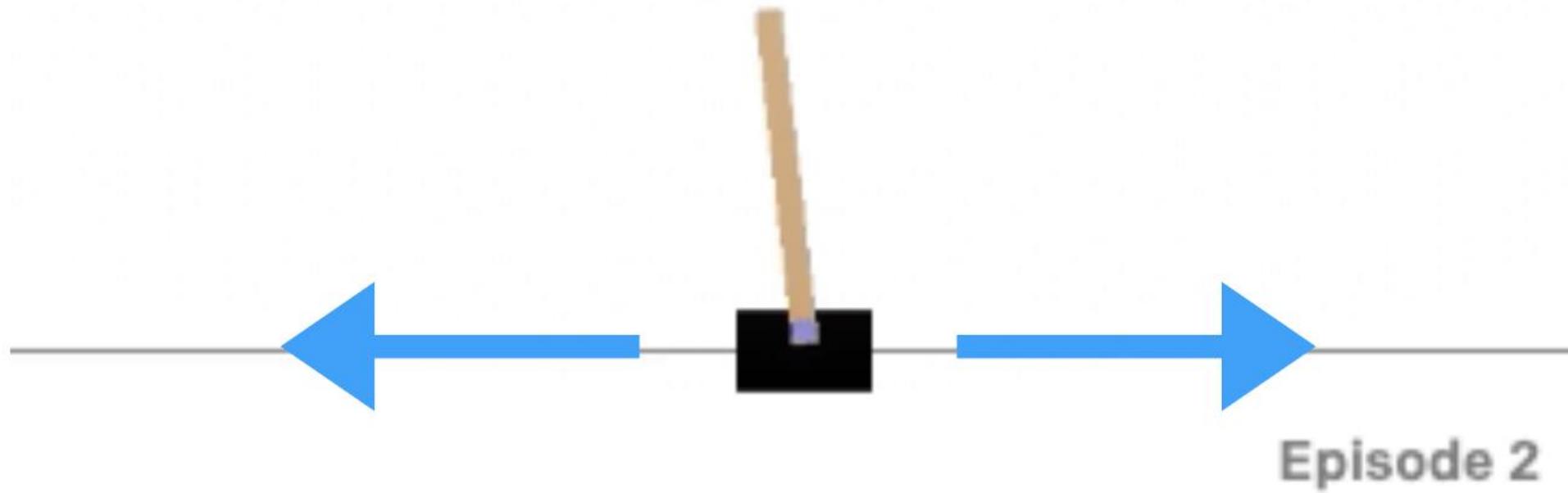
<https://wayve.ai/blog/learning-to-drive-in-a-day-with-reinforcement-learning/>

Reinforcement Learning for Robotics

➤ Optimal Control



Cart-Pole environment

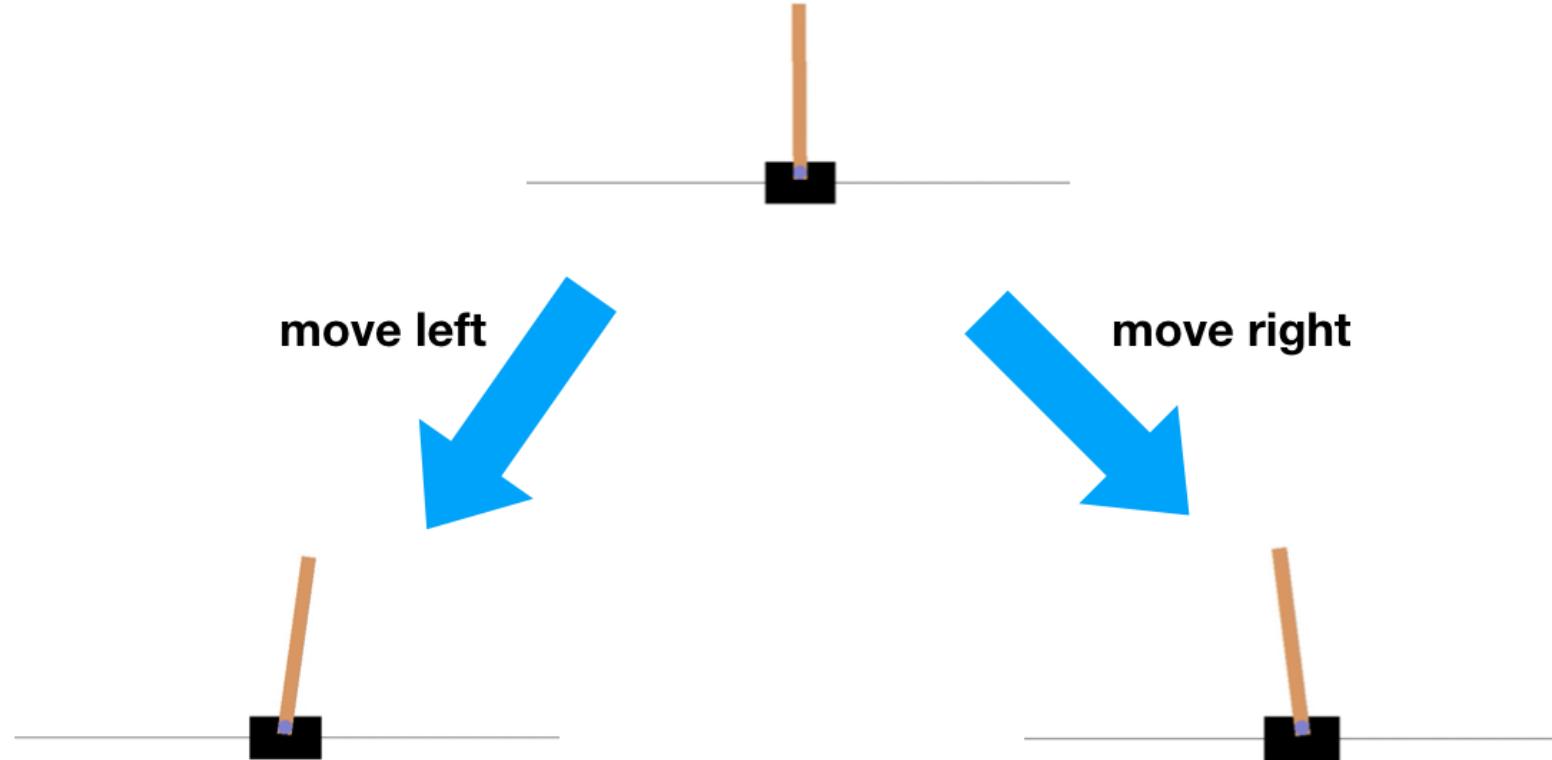


State yang tersedia dari env

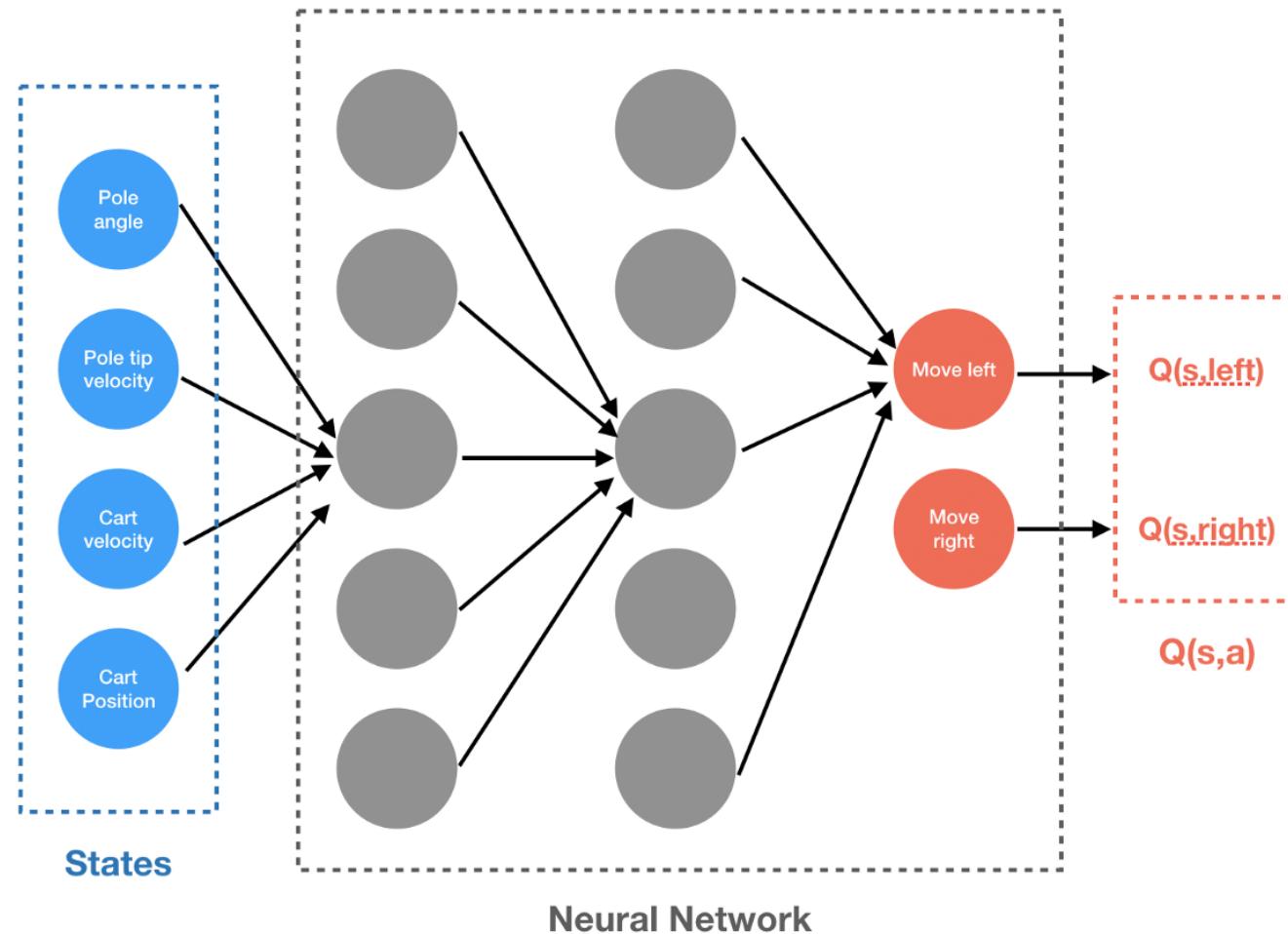
Num	Observation	Min	Max
0	Posisi Cart	-4.8	4.8
1	Kecepatan Cart	-inf	inf
2	Sudut Pendulum	-0.418 r (-24°)	0.418 r (24°)
3	Kecepatan Sudut Pendulum	-inf	inf

- Untuk Posisi Cart yang dapat diamati meskipun tersedia rentang $-4.8^{\sim}4.8$ namun game akan berakhir saat posisi keluar dari rentang $-2.4^{\sim}2.4$
- Meskipun Sudut Pendulum yang dapat diamati tersedia rentang $-24^{\circ\sim}-24^{\circ}$ namun game akan berakhir saat posisi keluar dari rentang $-12^{\circ\sim}12^{\circ}$

Action yang tersedia pada env



Penerapan DQL pada Cart-Pole



Hasil Pembelajaran DQL pada cart pole



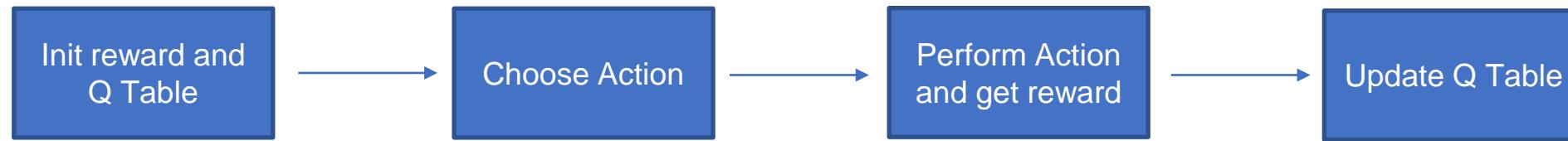


02

Static Goals Navigation using Reinforcement Learning

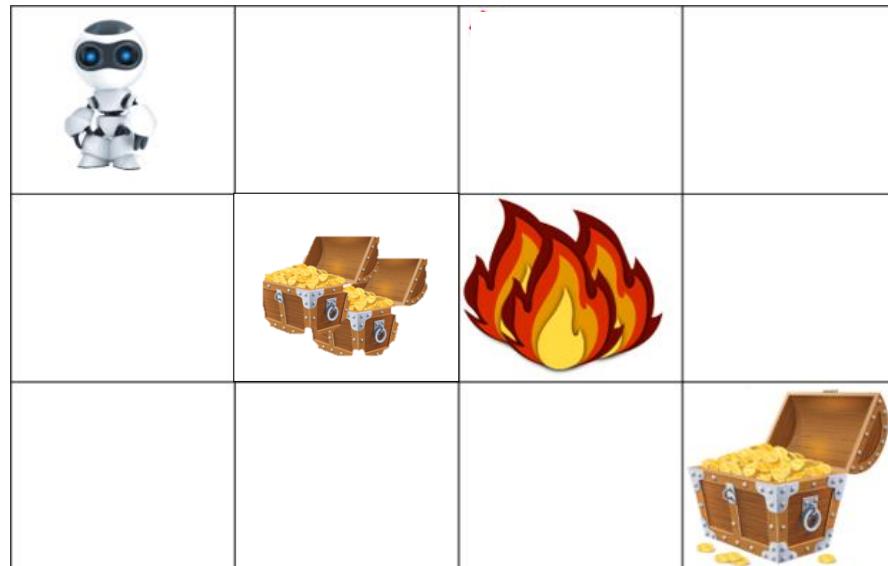
- Q Learning Implementation

How Q Learning work?

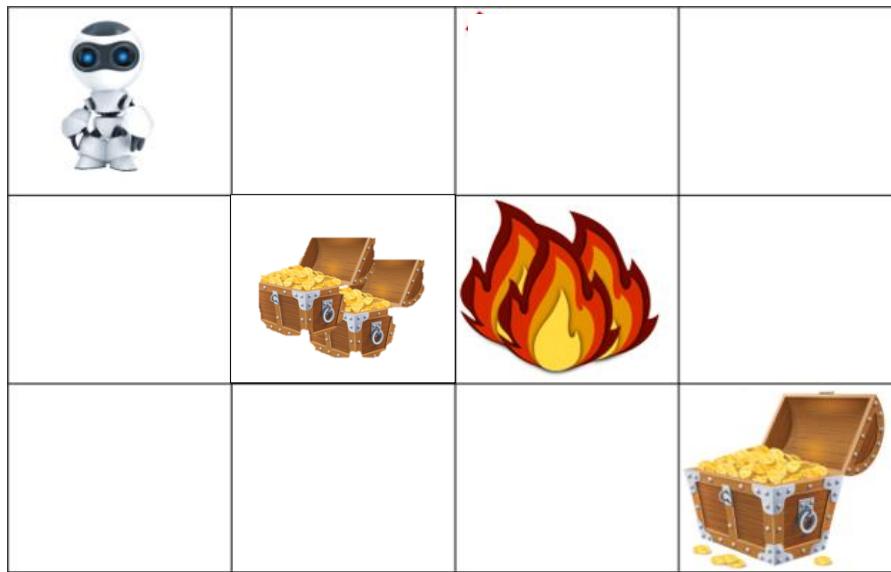


Studi Kasus

Studi kasus kali ini kita akan membantu robot Orbit untuk menemukan harta karun yang paling banyak dengan step yg sedikit dan menghindari supaya tidak terkena api azab.



1. Definisi Reward dan Q Table



Reward Table :

State	Reward
Empty	-1
Fire	-10
1 Harta Karun	+10
2 Harka Karun	+25

1. Definisi Reward dan Q Table (cont)



Actions

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

2. Choose Action

Pada dasarnya dalam algoritma Q-Learning, *agent* akan memilih *action* berdasarkan Q-Table. *Agent* akan memilih *action* yang mempunyai Q-Value paling besar berdasarkan *state* saat ini. Tapi.... Hal tersebut akan terdengar aneh untuk Langkah pertama, bukan? Karena pada Langkah pertama semua Q-Value yg ada pada Q-Table bernilai 0, jadi gimana cara *agent* memilih *action*?

Nah, untuk jawab pertanyaan ini kita akan coba mengingat Kembali tentang Eksplorasi dan Eksplotasi.

2. Choose Action : Eksplorasi vs Eksplotasi

Eksplorasi adalah suatu metode pemilihan *action* dimana *agent* akan melakukan pemilihan *action* secara *random* dengan tujuan ia bisa mengetahui informasi tentang *environment* secara mendalam.

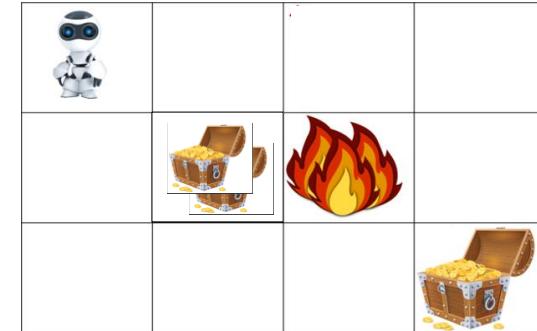
Sedangkan **Eksplotasi** adalah sebuah metode pemilihan *action* dengan memilih *action* yang mempunyai *return* (dalam hal ini Q-Value) paling besar.

Lalu, pasti teman teman semua berfikir kalau pemilihan *action* dengan metode eksplotasi adalah yg terbaik. Tentu saja tidak begituu! Loh kok gituu?? Kenapa???

2. Choose Action : Eksplorasi vs Eksplotasi (cont)

Mari kita bayangkan *environtment* kita adalah studi kasus kita pada kali ini.

Bagaimana kalau *agent* (robot orbit) menemukan terlebih dahulu satu harta karun, lalu kita menggunakan metode eksplotasi dalam memilih *action*? Maka yang terjadi adalah *agent* akan cenderung memilih *action* yg akan menuntun dia ke satu harta karun dibandingkan dua harta karun, bam!!

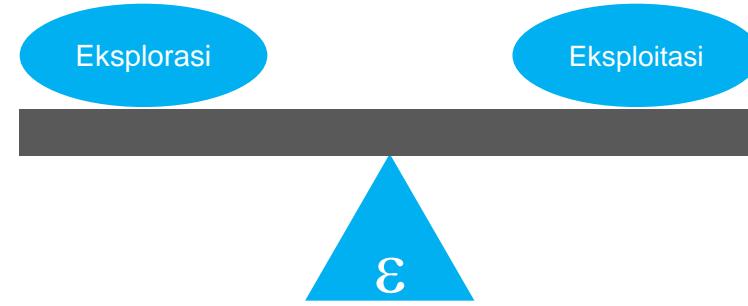


Lalu apakah metode eksplorasi adalah yg terbaik? Oh tentu tidak juga! Jika kita menggunakan metode eksplorasi (memilih *action* secara random) secara terus menerus maka kita akan kesulitan untuk mencapai solusi yang optimum.

Lalu solusinya bagaimana? Jawabannya adalah metode **epsilon-greedy exploration**

2. Choose Action : epsilon-greedy exploration

epsilon-greedy exploration digunakan untuk menyeimbangkan antara eksplorasi dan eksplotasi menggunakan nilai epsilon (ε). ε biasa disebut dengan rate eksplorasi.

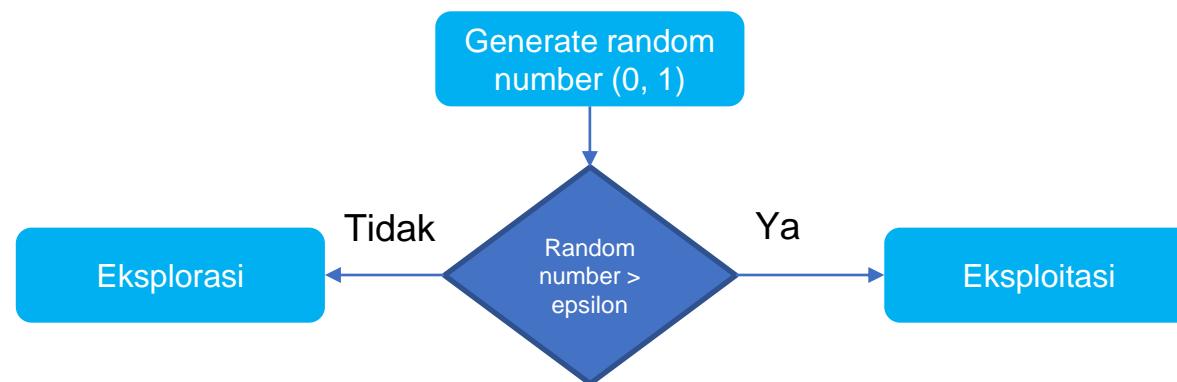


Pertama kita set nilai $\varepsilon = 1$, itu artinya pada episode – episode awal kita akan menggunakan metode eksplorasi dalam memilih *action* dengan tujuan agar *agent* dapat mengenali *environtment*-nya dengan baik.

2. Choose Action : epsilon-greedy exploration (cont)

Lalu dengan seiring berjalannya waktu nilai ϵ akan kita kurangi dengan rate tertentu, sehingga *agent* akan cenderung memilih *action* menggunakan metode Eksplorasi.

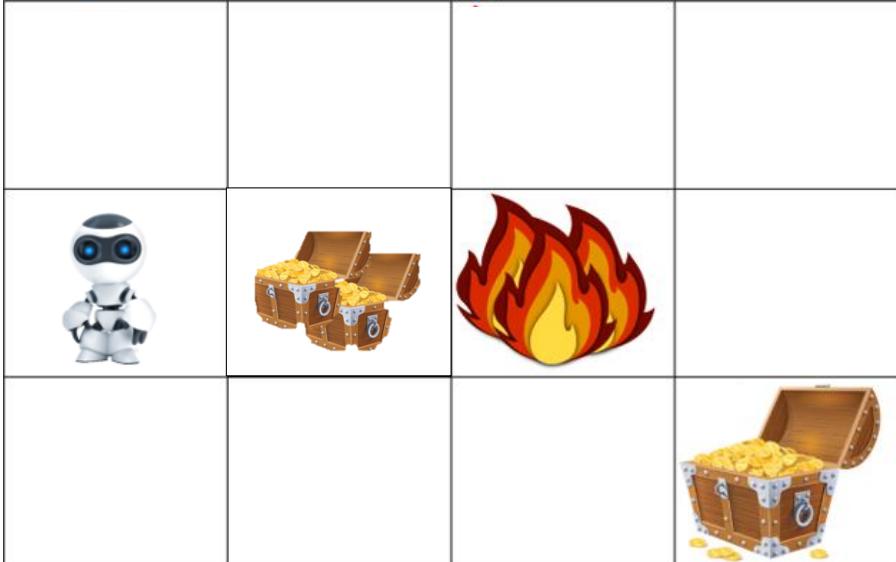
Lalu, untuk memberi tahu *agent* kapan harus pakai eksplorasi dan kapan harus pakai eksplloitasi, kita akan generate nomor acak dari 0 sampai 1. Jadi jika digambarkan dengan flowchart maka akan menjadi seperti ini.



2. Choose Action

Kembali ke Langkah kedua yaitu pemilihan *action*, maka sesuai dengan metode **epsilon-greedy exploration**, kita akan set nilai $\epsilon = 1$ dan untuk awal kita akan memilih *action* secara random. Kita misalkan *agent* memilih *action* ke bawah.

3. Perform Action and Get Reward



Reward = -1

4. Update Q-Table

Setelah eksekusi *action* dan mendapatkan *reward*, saatnya kita mengupdate Q-Table dengan mengganti Q-Value yg lama dengan Q-Value yang baru, untuk mengupdate Q-Table dan menghitung Q-Value yg baru kita dapat menggunakan formula berikut :

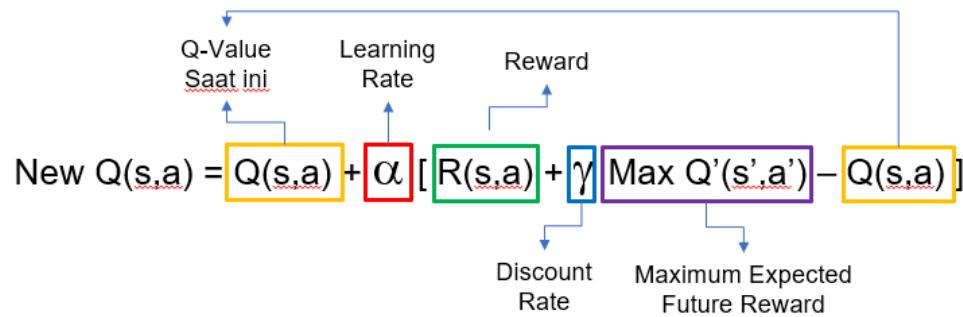
$$\text{New } Q(s,a) = Q(s,a) + \alpha [R(s,a) + \gamma \text{Max } Q'(s',a') - Q(s,a)]$$

The diagram illustrates the Q-learning update equation with arrows indicating the flow of information:

- A blue arrow points down to the term $Q(s,a)$.
- An arrow points up from the term α to the label "Learning Rate".
- An arrow points up from the term $R(s,a)$ to the label "Reward".
- An arrow points down from the term γ to the label "Discount Rate".
- An arrow points down from the term $\text{Max } Q'(s',a')$ to the label "Maximum Expected Future Reward".
- A blue arrow points down to the term $-Q(s,a)$.

4. Update Q-Table (cont)

Kita akan coba update Q-Table untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$:



$$\text{New } Q(\text{s},\text{a}) = Q(\text{s},\text{a}) + \alpha [R(\text{s},\text{a}) + \gamma \text{Max } Q'(\text{s}',\text{a}') - Q(\text{s},\text{a})]$$

$$\text{Max } Q'(\text{s}',\text{a}') = \text{Max}(q((2,1), \text{kiri}), q((2,1), \text{kanan}), q((2,1), \text{atas}), q((2,1), \text{bawah}))$$

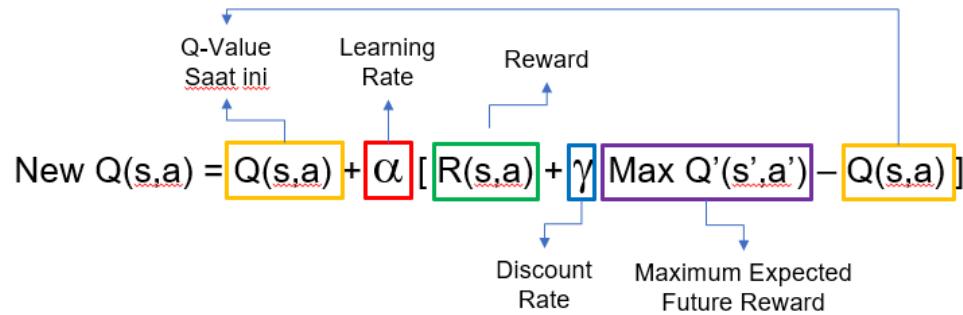
$$\text{Max } Q'(\text{s}',\text{a}') = \text{Max}(0, 0, 0, 0)$$

$$\text{Max } Q'(\text{s}',\text{a}') = 0$$

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

4. Update Q-Table (cont)

Kita akan coba update Q-Table untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$:



Max Q'(s',a') = 0

$$\text{New } Q(s,a) = Q(s,a) + \alpha [R(s,a) + \gamma \text{Max } Q'(s',a') - Q(s,a)]$$

$$\text{New } Q((1,1), \text{bawah}) = 0 + 0.7 [-1 + 0.99 (0) - 0]$$

$$\text{New } Q((1,1), \text{bawah}) = 0 + 0.7 [-1]$$

$$\text{New } Q((1,1), \text{bawah}) = -0.7$$

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

4. Update Q-Table (cont)

Kita akan coba update Q-Table untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$:

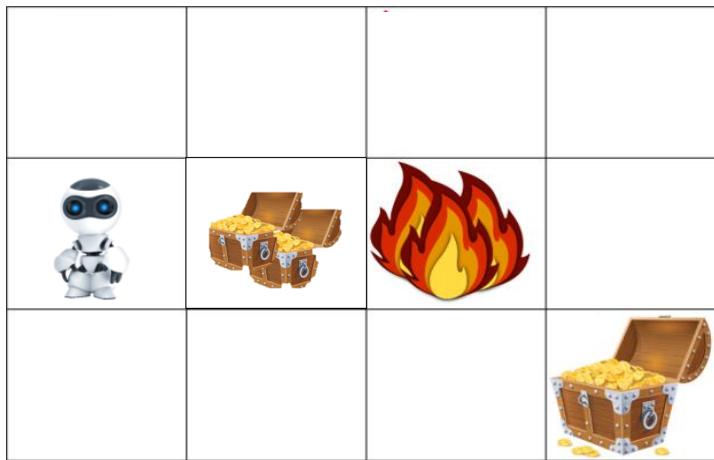
$$\text{Max } Q'(s',a') = 0$$

$$\text{New } Q(s,a) = Q(s,a) + \alpha [R(s,a) + \gamma \text{ Max } Q'(s',a') - Q(s,a)]$$

$$\text{New } Q((1,1), \text{ bawah}) = 0 + 0.7 [-1 + 0.99 (0) - 0]$$

$$\text{New } Q((1,1), \text{ bawah}) = 0 + 0.7 [-1]$$

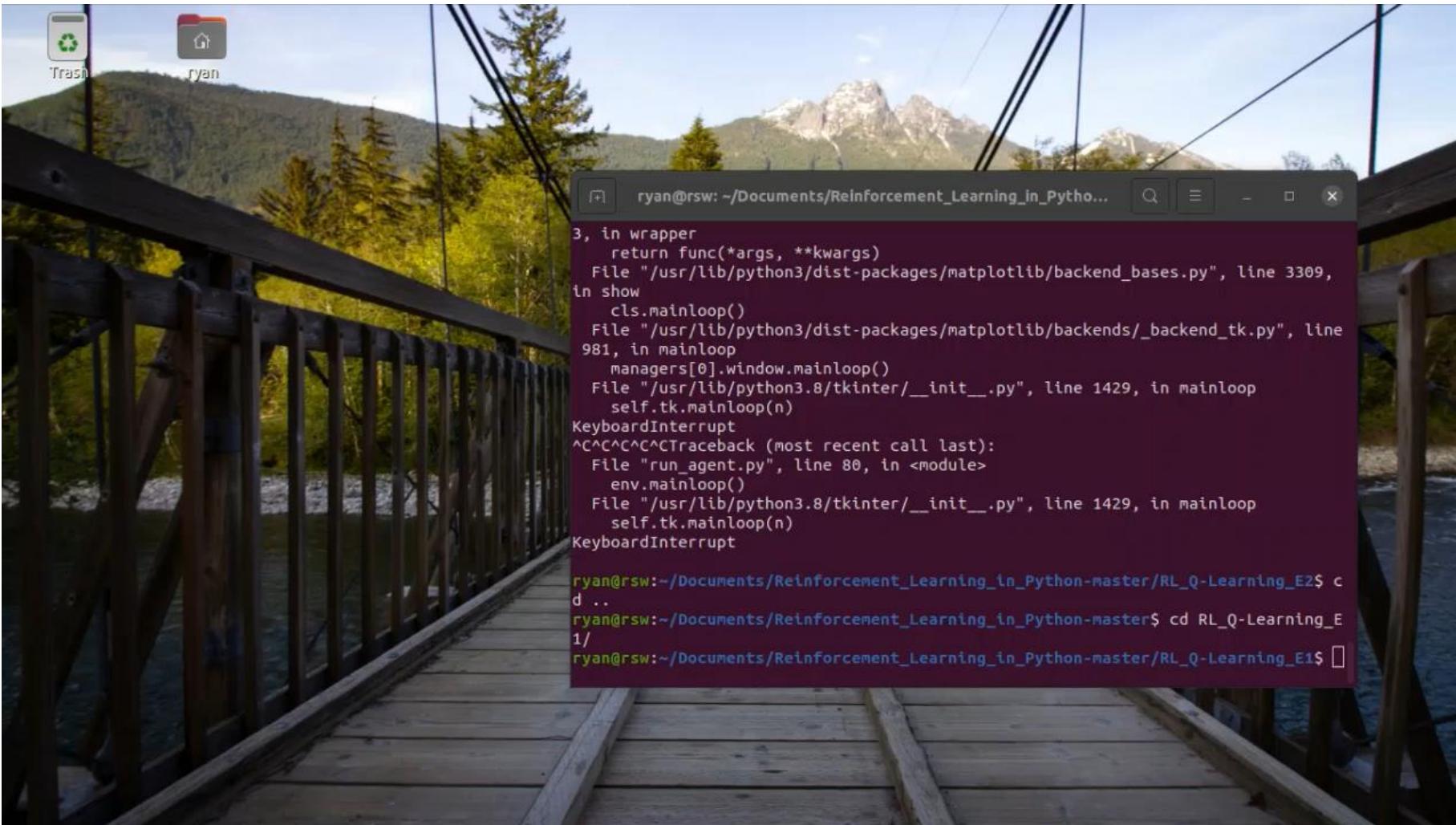
$$\text{New } Q((1,1), \text{ bawah}) = -0.7$$



	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	-0.7
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

Q-Learning Implementation

- On-policy:



Q-Learning Implementation

- Student Activity I
 - Buat kelompok yang terdiri dari 5 orang
 - Download code yang ada di drive
 - Terdapat 3 file dan 1 folder image, 3 file tersebut adalah agent_brain.py berisikan tentang algoritma QL, env.py berisikan tentang cara membuat environment yang mengambil input gambar dari folder image, dan run_agent.py yg berisikan untuk running agent.
 - Untuk menjalankan program, tinggal jalankan run_agent.py saja.
 - Tugas : merubah tata letak environment yg sudah ada, mulai dari posisi robot, posisi goal yang dituju, dan tata letak obstacle. Rubah di bagian env.py

- Code in Drive link:

https://ptorbitventurainodnesia-my.sharepoint.com/:f/g/personal/ryan_orbitfutureacademy_sch_id/EvGMVJnTVY1FnBzGvWwY4SIBd30m1LhJ36Jk3Ychbl3Qow?e=nYEFav



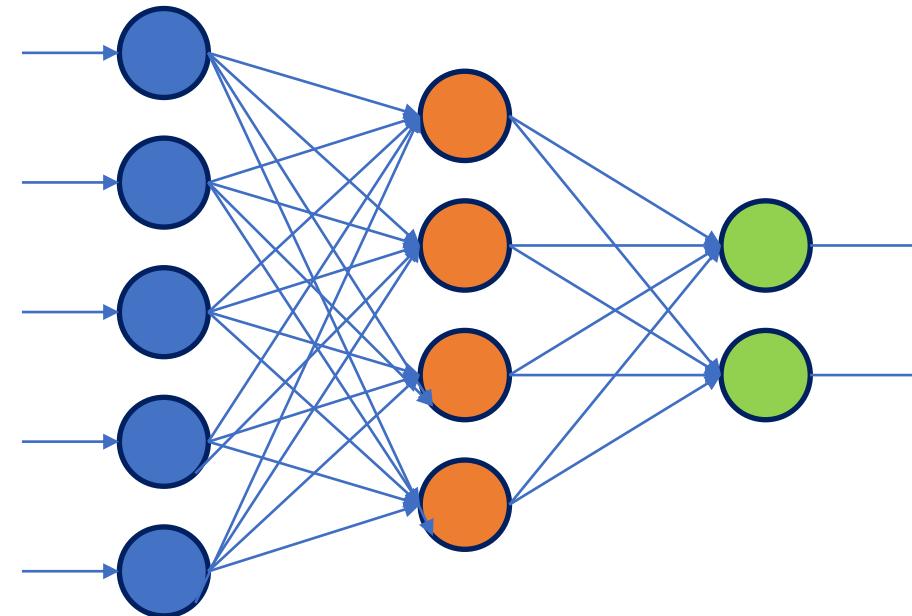
03

Dynamic Goals Navigation using Reinforcement Learning

- Deep Q Learning Implementation

Deep Q Learning

Pada *Deep Q Learning* kita akan menggantikan Q-Table menggunakan sebuah *Neural Network* yang biasa disebut dengan *Deep Q Network* atau DQN.



Deep Q Network

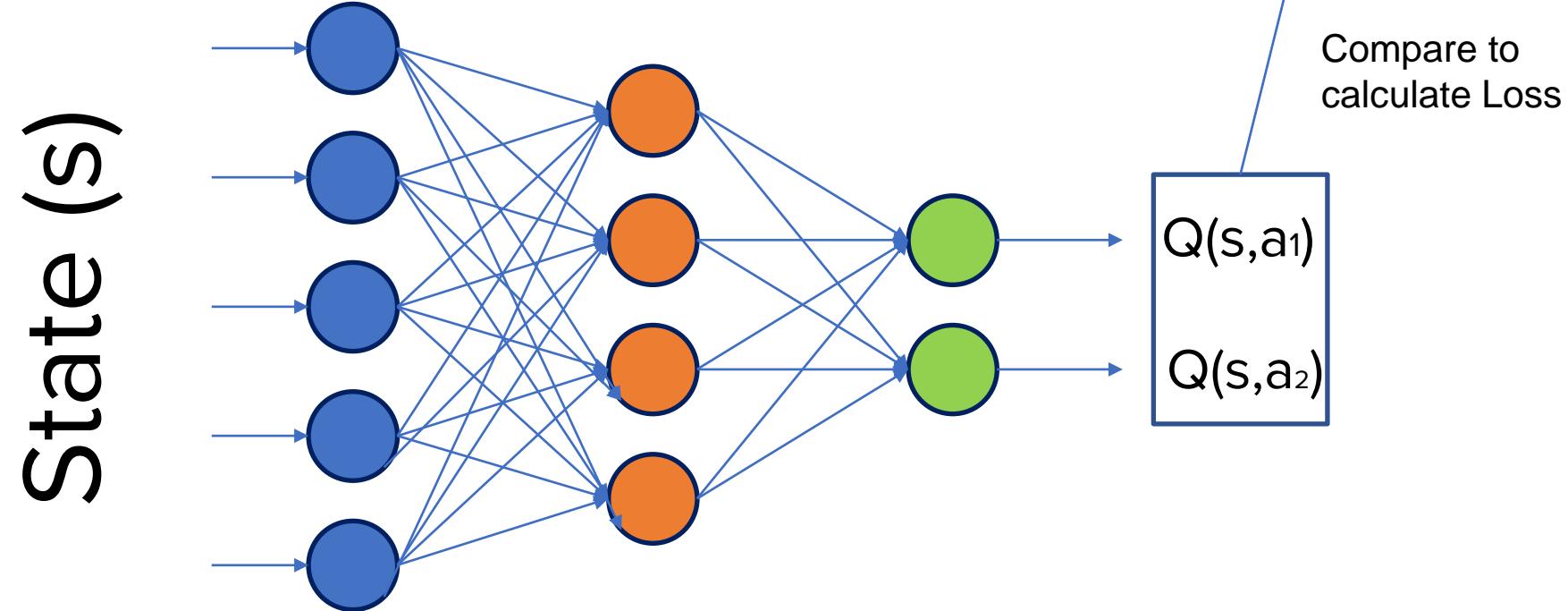
Deep Q Network adalah sebuah NN yang menerima *states* yg diberikan oleh *environment* sebagai input, lalu DQN akan menghasilkan output estimasi Q Values pada setiap *actions* yang dapat diambil pada *state* tersebut. Tujuan dari NN ini adalah untuk menghasilkan aproksimasi Q Function yg optimal. Seperti yg kita tau pada slide sebelumnya kalau Optimal Q Function adalah seperti ini :

$$q_* (s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_* (s', a') \right]$$

Loss pada NN ini adalah dengan membandingkan antara Q-Values dari output dengan target Q-Values yang didapatkan dari persamaan diatas

Deep Q Network (cont)

$$q_*(s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') \right]$$



Goal dari NN ini adalah untuk minimize loss, lalu setelah menghitung loss bobot pada *network* akan diupdate menggunakan *stochastic gradient descent* dan *backpropagation* seperti *neural network* pada umumnya.

Experience Replay and Replay Memory

Dalam proses *training* DQN kita akan menggunakan sebuah teknik yg dinamakan dengan ***experience replay***. Dengan *experience replay*, kita menyimpan *experience* dari *agent* untuk setiap *time step* ke dalam sebuah wadah yang bernama ***replay memory***.

Kita dapat merepresentasikan *experience* dari *agent* untuk setiap t sebagai e_t . Pada time t, *agent experience* e_t didefinisikan sebagai tuple berikut :

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

Tuple ini menyimpang *state* (s_t), *action* (a_t) yang diambil, reward (r_{t+1}) yang didapatkan oleh *agent* dan *state* selanjutnya (s_{t+1}).

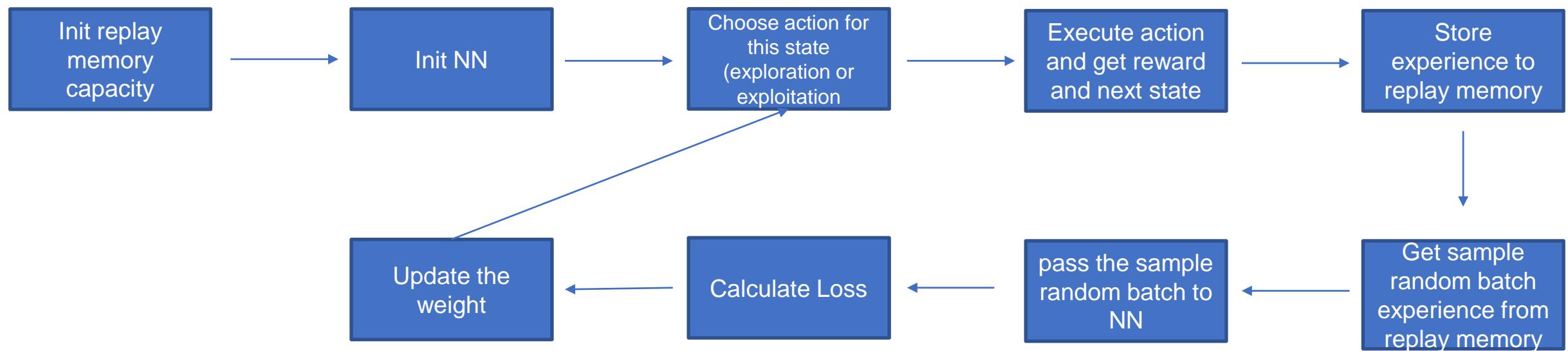
Experience Replay and Replay Memory (cont)



Secara teori seluruh *experience agent* pada setiap *time step* akan disimpan pada *replay memory*. Sebenarnya dalam praktiknya, kita akan mendefinisikan besaran *experience* yang dapat ditampung oleh *replay memory* sebesar N, dan kita hanya akan menyimpan N terakhir *experience* dari *agent*.

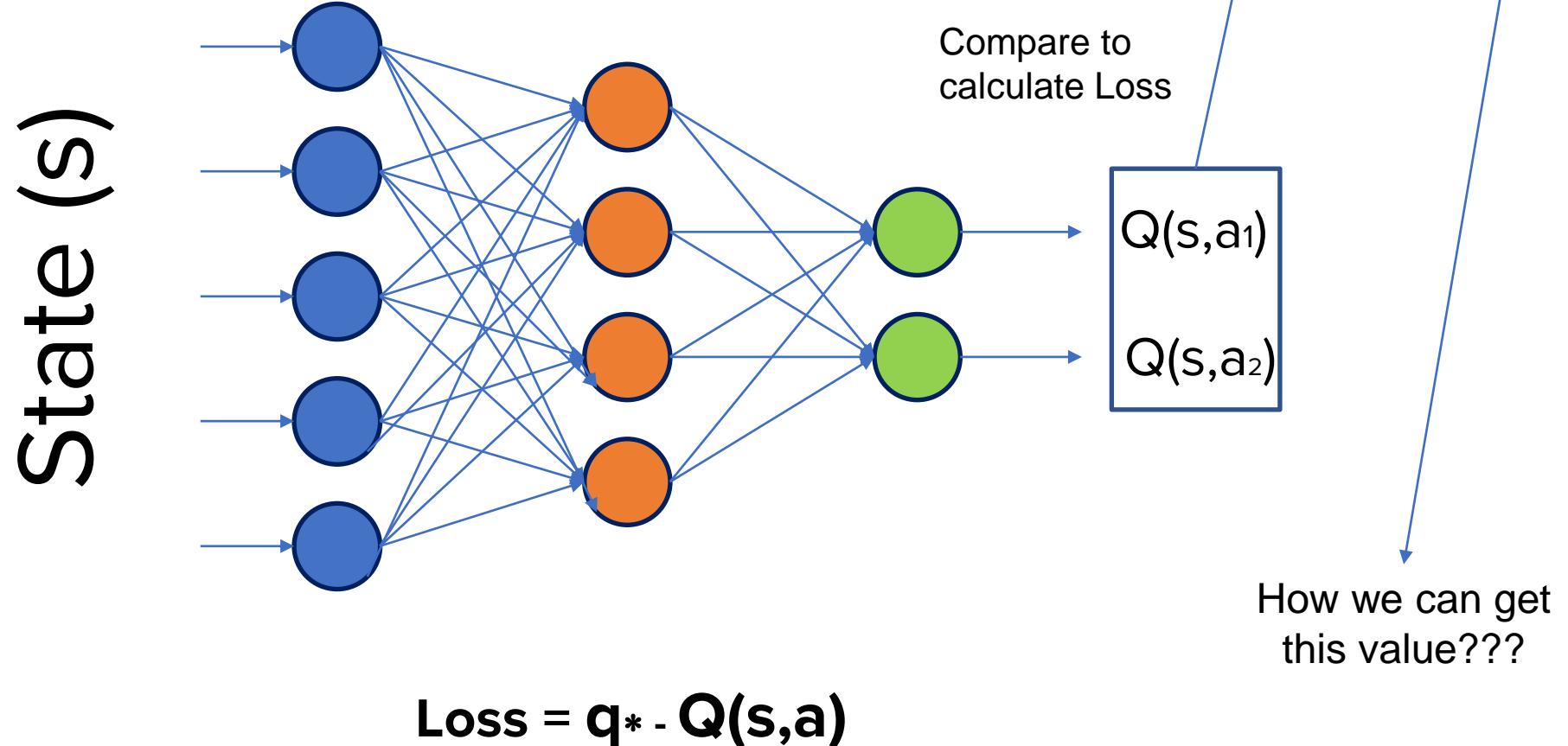
Source gambar : deeplizard

How we Train the DQN?



How We Calculate Loss on DQN?

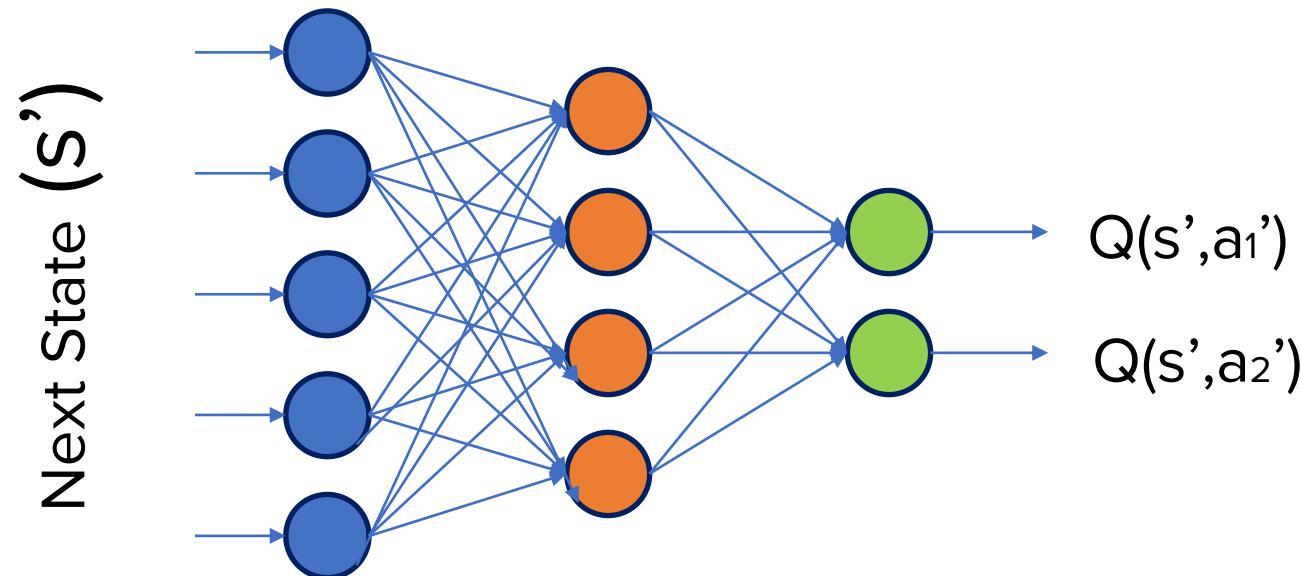
$$q_*(s, a) = E \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') \right]$$



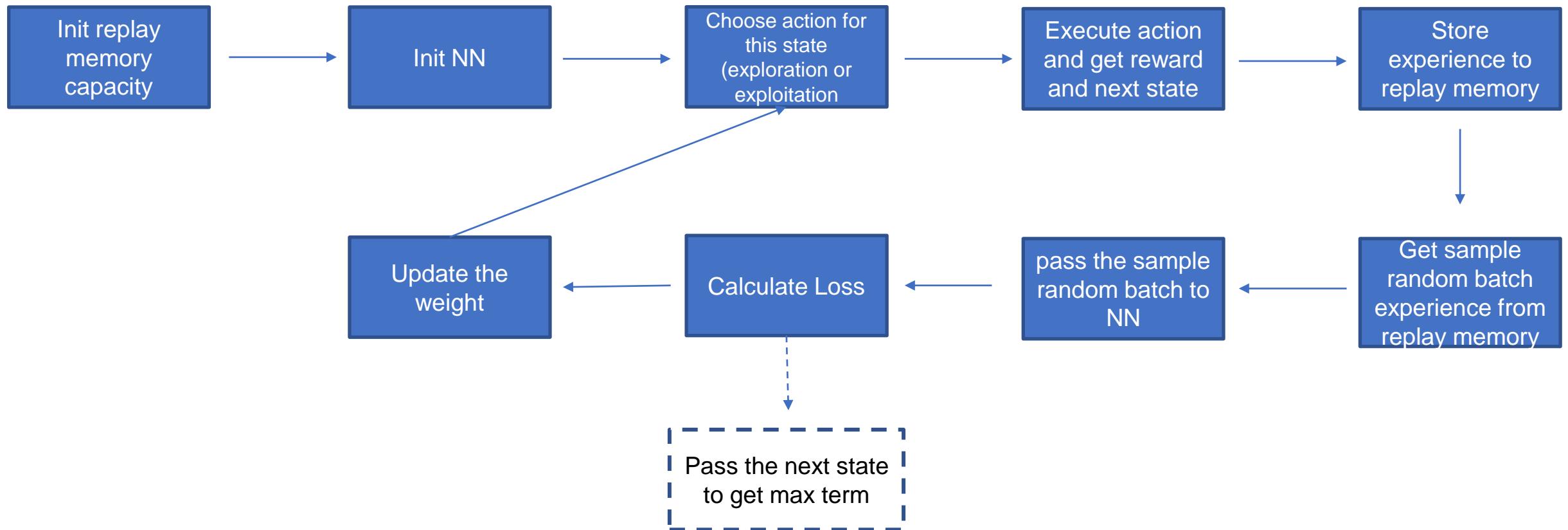
How We Calculate Loss on DQN? : Calculate the Max Term

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

Pada Q Learning kita dapat mendapatkan value diatas dengan melihatnya di Q-Table, tapi itu adalah cara yg kunooo! Pada Deep Q Learning kita akan mendapatkan value tersebut dengan bantuan DQN kita



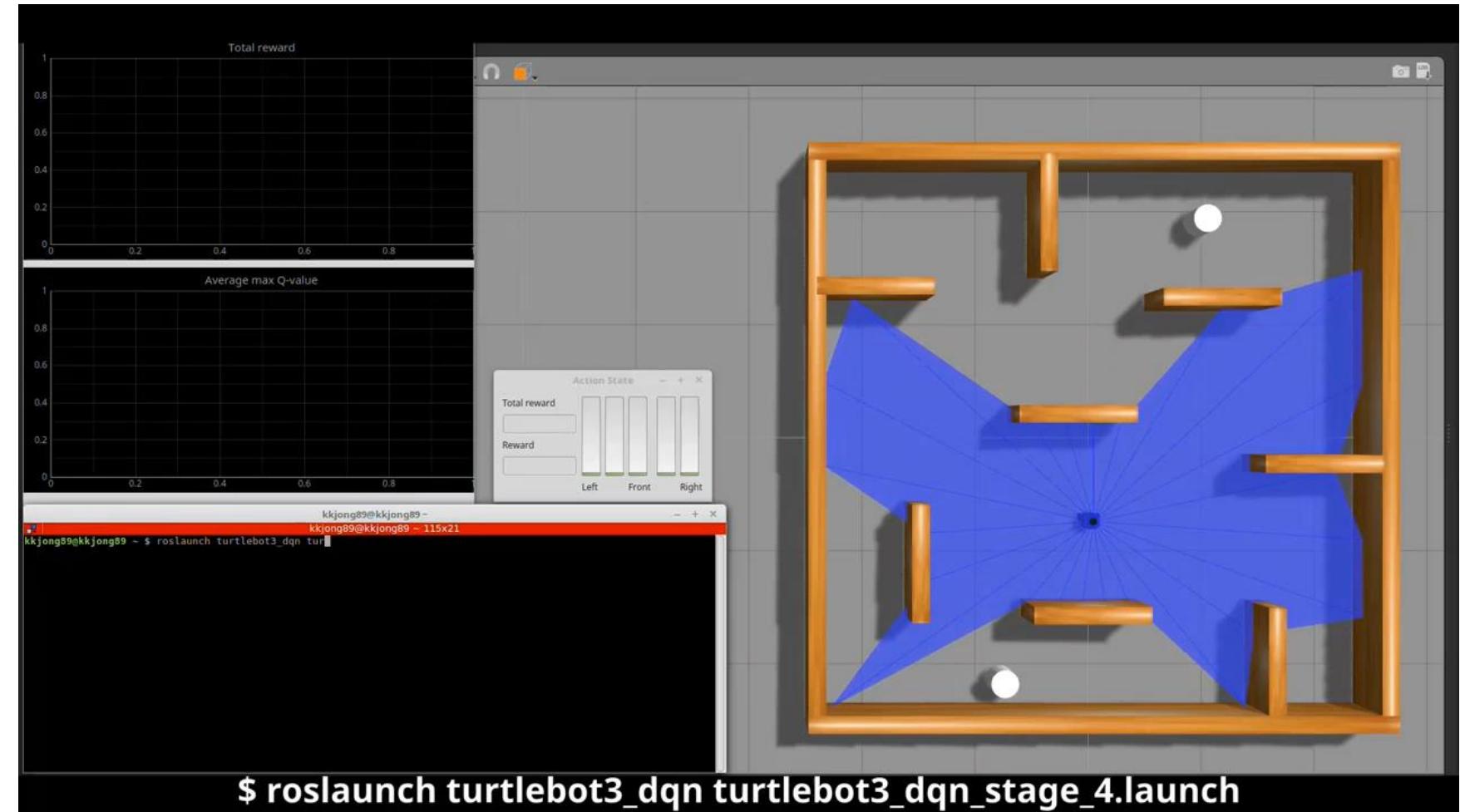
How to train DQN



Deep Q-Learning Implementation

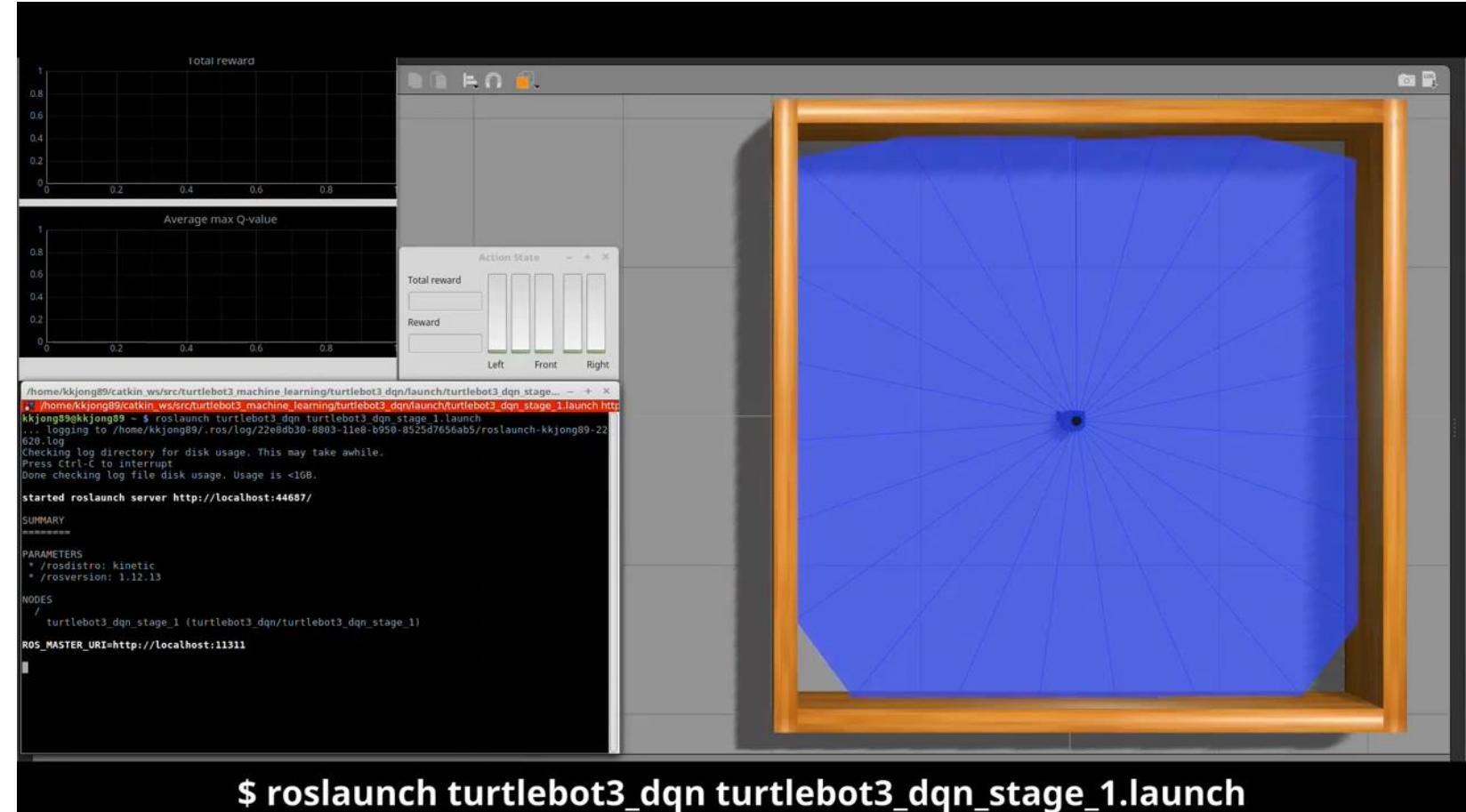
Indoor Dynamic goals Navigation using Reinforcement Learning.

There are a static obstacle and dynamic obstacle (white tabular). Imagine the static obstacle is a wall and dynamic obstacle is human.



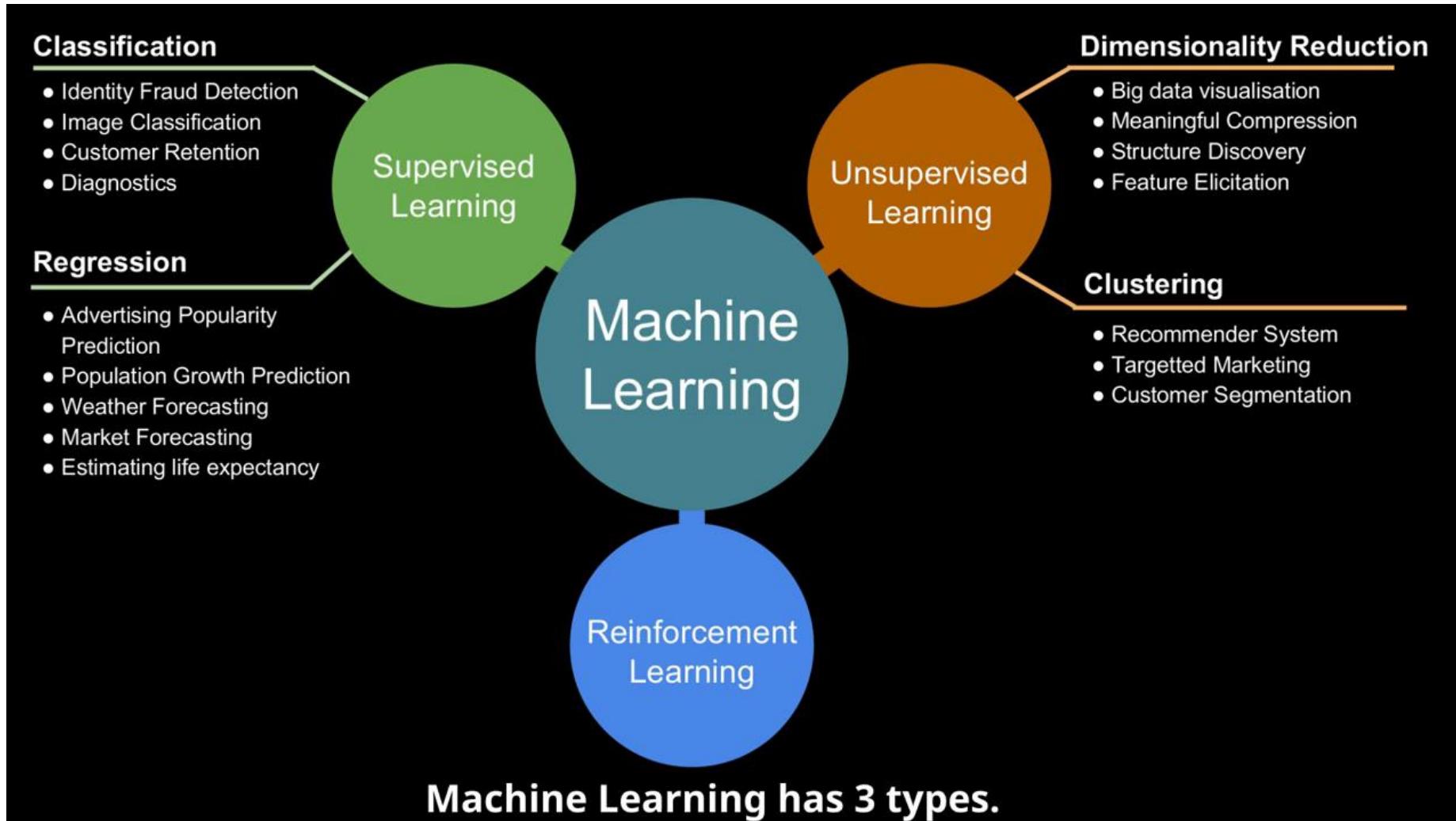
Deep Q-Learning Implementation

The simplest case of indoor dynamic goal navigation without any dynamic obstacle and less wall.



Deep Q-Learning Implementation

How to build this model ?



Turtlebot3 using pygame version

- Define Q Trainer Part I
1. Creating a class named Linear_Qnet for initializing the linear neural network.
 2. The function forward is used to take the input(11 state vector) and pass it through the Neural network and apply relu activation function and give the output back i.e the next move of 1×3 vector size. In short, this is the prediction function that would be called by the agent.
 3. The save function is used to save the trained model for future use.

```
class Linear_QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x

    def save(self, file_name='model_name.pth'):
        model_folder_path = 'Path'
        file_name = os.path.join(model_folder_path, file_name)
        torch.save(self.state_dict(), file_name)
```

Deep Q-Learning Implementation

▪ Define Q Trainer Part II

1. Initialising QTrainer class

- setting the learning rate for the optimizer.
- Gamma value that is the discount rate used in Bellman equation.
- initialising the Adam optimizer for updation of weight and biases.
- criterion is the Mean squared loss function.

2. Train_step function

- As you know that PyTorch work only on tensors, so we are converting all the input to tensors.
- As discussed above we had a short memory training then we would only pass one value of state, action, reward, move so we need to convert them into a vector, so we had used unsqueezed function .
- Get the state from the model and calculate the new Q value using the below formula:

$$Q_{\text{new}} = \text{reward} + \gamma * \max(\text{next_predicted Qvalue})$$

- calculate the mean squared error between the new Q value and previous Q value and backpropagate that loss for weight updation.

```
class QTrainer:  
    def __init__(self,model,lr,gamma):  
        #Learning Rate for Optimizer  
        self.lr = lr  
        #Discount Rate  
        self.gamma = gamma  
        #Linear NN defined above.  
        self.model = model  
        #optimizer for weight and biases updation  
        self.optimizer = optim.Adam(model.parameters(),lr = self.lr)  
        #Mean Squared error loss function  
        self.criterion = nn.MSELoss()  
  
  
    def train_step(self,state,action,reward,next_state,done):  
        state = torch.tensor(state,dtype=torch.float)  
        next_state = torch.tensor(next_state,dtype=torch.float)  
        action = torch.tensor(action,dtype=torch.long)  
        reward = torch.tensor(reward,dtype=torch.float)  
  
        #only one parameter to train,  
        # Hence convert to tuple of shape(1, x)  
        if(len(state.shape) == 1):  
            #(1, x)  
            state = torch.unsqueeze(state,0)  
            next_state = torch.unsqueeze(next_state,0)  
            action = torch.unsqueeze(action,0)  
            reward = torch.unsqueeze(reward,0)  
            done = (done, )  
  
        # 1. Predicted Q value with current state  
        pred = self.model(state)  
        target = pred.clone()  
        for idx in range(len(done)):  
            Q_new = reward[idx]  
            if not done[idx]:  
                Q_new = reward[idx] +  
                    self.gamma * torch.max(self.model(next_state[idx]))  
            target[idx][torch.argmax(action).item()] = Q_new  
        # 2. Q_new = reward + gamma * max(next_predicted Qvalue)  
        #pred.clone()  
        #preds[argmax(action)] = Q_new  
        self.optimizer.zero_grad()  
        loss = self.criterion(target,pred)  
        loss.backward() # backward propagation of loss  
  
        self.optimizer.step()
```

Turtlebot3 using pygame version

- Define the agent part I

Define the state (11 states)

```
state = [
    # Danger straight
    (dir_r and game.is_collision(point_r)) or
    (dir_l and game.is_collision(point_l)) or
    (dir_u and game.is_collision(point_u)) or
    (dir_d and game.is_collision(point_d)),

    # Danger right
    (dir_u and game.is_collision(point_r)) or
    (dir_d and game.is_collision(point_l)) or
    (dir_l and game.is_collision(point_u)) or
    (dir_r and game.is_collision(point_d)),

    # Danger left
    (dir_d and game.is_collision(point_r)) or
    (dir_u and game.is_collision(point_l)) or
    (dir_r and game.is_collision(point_u)) or
    (dir_l and game.is_collision(point_d)),

    # Move direction
    dir_l,
    dir_r,
    dir_u,
    dir_d,

    # Food location
    game.food.x < game.head.x, # food left
    game.food.x > game.head.x, # food right
    game.food.y < game.head.y, # food up
    game.food.y > game.head.y # food down
]
print(state[0],state[1],state[2],state[3],state[4],state[5],state[6],state[7],state[8],state[9],state[10])
```

Define the action
(3 actions : Straight, Turn right, Turn left)

```
def _move(self, action):
    """
    The Snake Can only move [Straight, Right, Left]
    """

    clock_wise = [Direction.RIGHT, Direction.DOWN, Direction.LEFT, Direction.UP]
    idx = clock_wise.index(self.direction)

    if np.array_equal(action, [1, 0, 0]):
        new_dir = clock_wise[idx] # no change
    elif np.array_equal(action, [0, 1, 0]):
        next_idx = (idx + 1) % 4
        new_dir = clock_wise[next_idx] # right turn r -> d -> l -> u
    else: # [0, 0, 1]
        next_idx = (idx - 1) % 4
        new_dir = clock_wise[next_idx] # left turn r -> u -> l -> d

    self.direction = new_dir

    x = self.head.x
    y = self.head.y
    if self.direction == Direction.RIGHT:
        x += BLOCK_SIZE
    elif self.direction == Direction.LEFT:
        x -= BLOCK_SIZE
    elif self.direction == Direction.DOWN:
        y += BLOCK_SIZE
    elif self.direction == Direction.UP:
        y -= BLOCK_SIZE

    self.head = Point(x, y)
```

Turtlebot3 using pygame version

- Define the agent part II

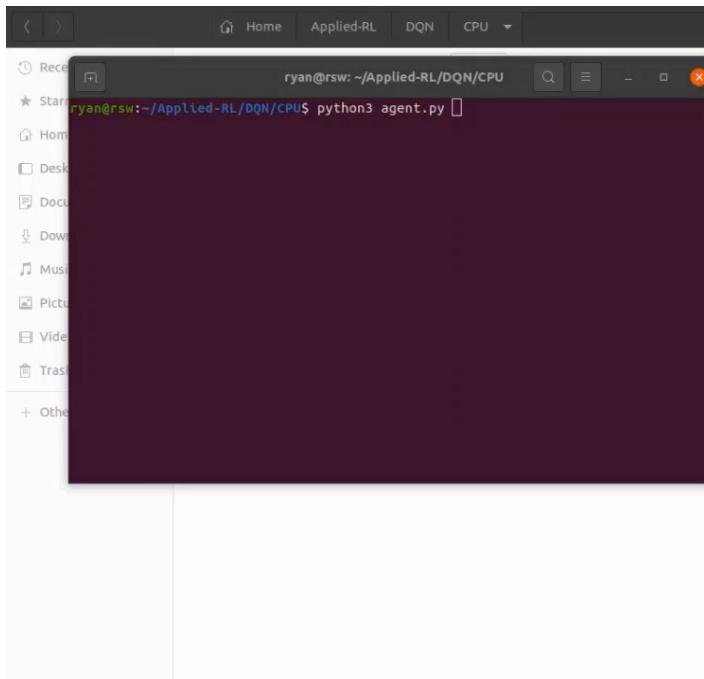
```
def get_action(self, state):
    # random moves: tradeoff exploration / exploitation
    self.epsilon = 80 - self.n_games
    final_move = [0,0,0]
    if random.randint(0, 200) < self.epsilon:
        move = random.randint(0, 2)
        final_move[move] = 1
    else:
        state0 = torch.tensor(state, dtype=torch.float)
        prediction = self.model(state0)
        move = torch.argmax(prediction).item()
        final_move[move] = 1

    return final_move
```

- Define the gradient policy to determine whether to be exploration or exploitation.

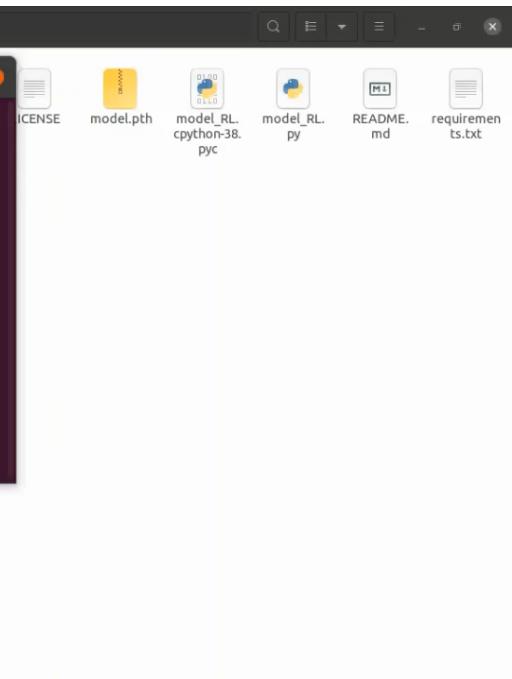
Turtlebot3 using pygame version

➤ Episode 1 ~ 10



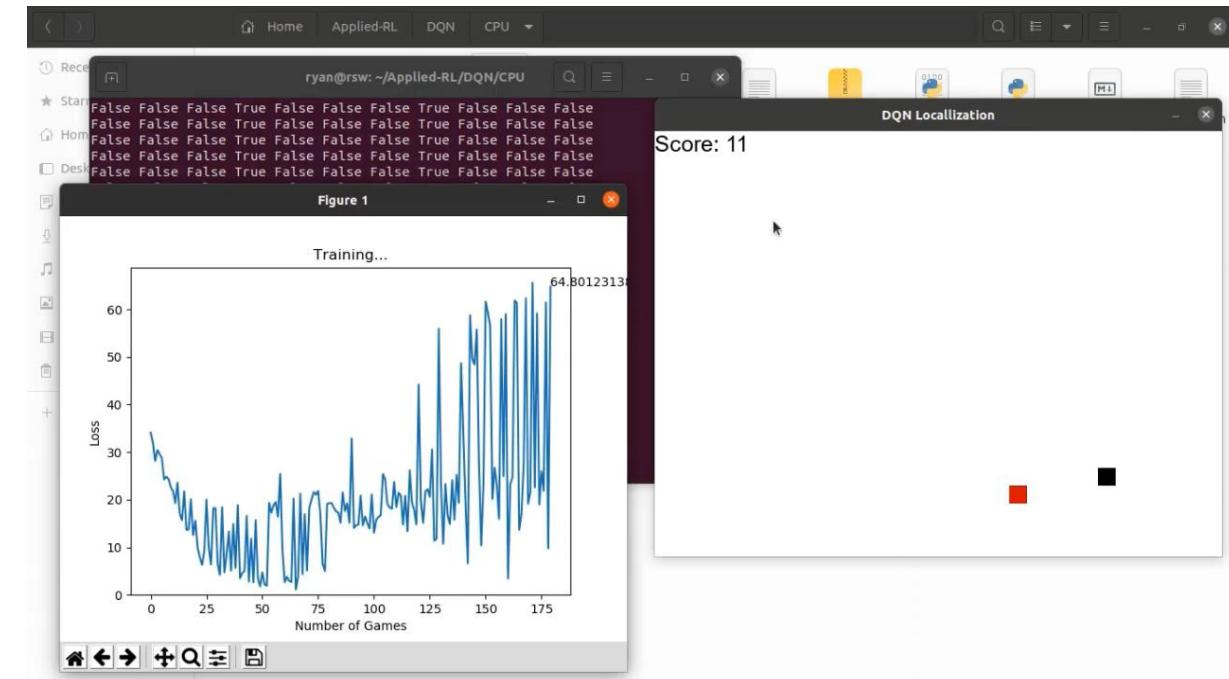
A terminal window titled "ryan@rsw: ~/Applied-RL/DQN/CPU\$". It displays the command "python3 agent.py" being run. The terminal window has a dark background and white text.

ryan@rsw:~/Applied-RL/DQN/CPU\$ python3 agent.py



A file manager window showing the contents of the "DQN" directory. It includes files: LICENSE, model.pth, model_RL_cython-38.pyc, model_RL_py, README.md, and requirements.txt.

➤ After training episode 175



Deep Q-Learning Implementation

- Student Activity II
 - Tugas Individu
 - Download code yang ada di drive
 - Untuk menjalankan program, tinggal jalankan agent.py saja.
 - Install beberapa library dan dependencies yang diperlukan.
 - Tugas : cobalah untuk menjalankan program DQL for indoor dynamic goal Navigation.
- Code in Drive link:
https://ptorbitventurainodnesia-my.sharepoint.com/:f/g/personal/ryan_orbitfutureacademy_sch_id/EgUJfYdyfpVPIwjDoXBt0IBgk11MVe0cIR_XYVWN0WNdQ?e=rRu4dN

References

- Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction. 2020
- Sudharsan Ravichandiran. Hands-On Reinforcement Learning with Python. 2018



This presentation is made by



Wayan Dadang

Wijaya Yudha Atmaja

Dino Febriyanto S.Kom.

Ryan Satria Wijaya S.T.

Oetomo





TERIMA KASIH

Orbit Future Academy

PT Orbit Ventura Indonesia
Center of Excellence (Jakarta Selatan)
Gedung Veteran RI, Lt.15
Unit Z15-002, Plaza Semanggi
Jl. Jenderal Sudirman Kav.50, Jakarta
12930, Indonesia

- Jakarta Selatan/Pusat
- Jakarta Barat/BSD
- Kota Bandung
- Kab. Bandung
- Jawa Barat

Hubungi Kami

Director of Sales & Partnership
ira@orbitventura.com
+62 858-9187-7388

Social Media

-  Orbit Future Academy
-  @OrbitFutureAcademyIn1
-  OrbitFutureAcademy
-  Orbit Future Academy