Reinforcement Learning (RL)
(A practical approach)

Chapter 6:

Q-Learning, SARSA, E-SARSA

Algorithms

Saeed Saeedvand, Ph.D.

Contents

- **Section one**
  - Reinforcement Learning (Review)
- **Section two**
  - Q-Learning
  - A simple example with Q-Learning
  - SARSA
  - E-SARSA
  - OpenAI Gym toolkit
- **Section three**
  - A paper that presented with Q-Learning
- **Section four**
  - Conclusion
- **Implementation**
  - Empirical example in python and gym toolkit, (Q-learning, SARSA, ESARSA)

Section one: Reinforcement Learning

Reinforcement Learning (RL)

✓ An interactive method and differs from supervised learning algorithms.

All supervised learning methods such as:
Random forest, support-vector machine, Bayesian, decision tree, ANN, and etc.

Train Phase

Creating a model based on the provided data

Test phase

Classifies what is learned in the train phase
+ some new data

But in RL, the aim is a learning from interactive environment, so that, the agent must learn actions itself.

So, it's not classification nor clustering

Reinforcement Learning Components:

1. **Policy**
   - ✓ Defines the agent's behavior over time. That is, policy says which action is best in each case.
2. **Reward function**
   - ✓ Specifies the **target** in the reward function.
   - ✓ The reward function is to provoke the agent for doing actions so that increases the reward.

Bad reward function = the agent learns late

Reinforcement Learning

3. Value function

- ✓ Evaluation of how good each state is
- ✓ Has a **long-term look**.
  - ➢ It's not like you're rewarded in one game for letting your opponent beat you, but you **go for something better in future**.

4. Model

- ✓ The agent **can go to the next state** for **one action**.
- ✓ At first we do not know what is the value of **going from one state to another**, (action it is good or bad).
- ✓ **Every action** is **one possibility**.
- ✓ Environment is **stochastic** and states are **nondeterministic**.
- ✓ The **goal** of the learner is to **maximize long-term rewards**.

Reinforcement Learning

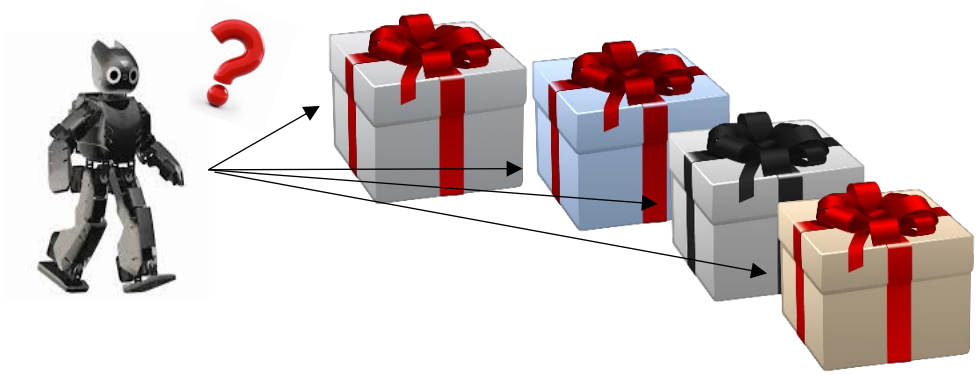
Tip:

- ✓ Reinforcement learning is therefore considered as a way to train agents to perform an action through **reward** and **punishment** without specifying how the agent performs the possible actions.

Reinforcement Learning

Action and reward: (other differences with supervised learning)

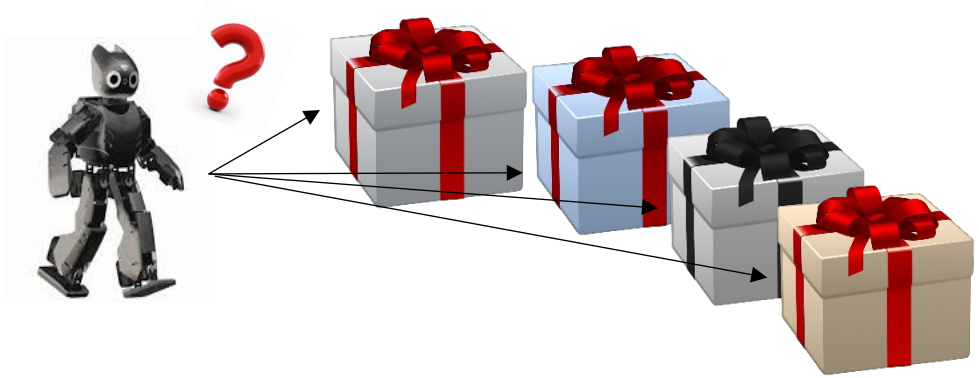
- ✓ The learning **examples** are **not** presented in pairs (**input/output**), but **receive a reward and move on to the next step after the action is done**.
- ✓ The agent has **no idea** what is the **best thing to do in each case**. (Agent must find himself).



- ✓ In unknown territories where learning is expected to be useful, the **agent must be able to learn from his experience**.

Reinforcement Learning

How to select (complementary notes)?



Model (states)?

series of actions? Or single action?

Policy (Actions)?

Again, series of actions? Or single action?

Reward function?

Value function?

10pc = 1000$

Reinforcement Learning

Exploration and exploitation (reminder)

One of the challenges in reinforcement learning:

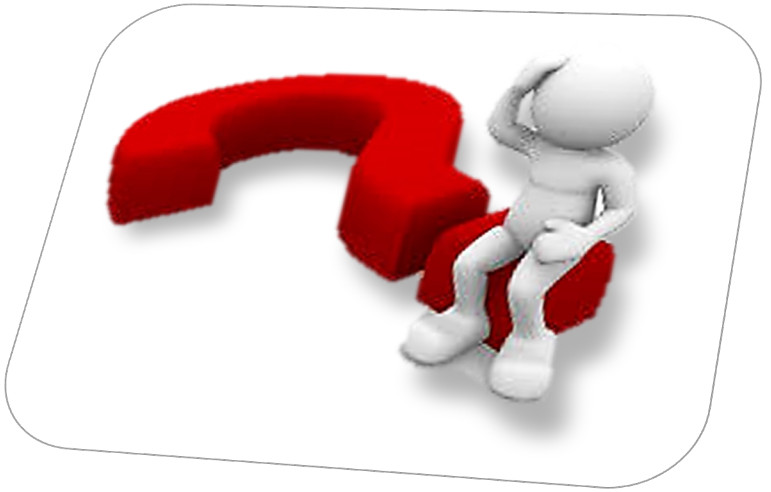
- ✓ Trade off or the interaction between **exploration** and **exploitation**.
- ✓ **Exploration** of new things means let the agent discover and learn.
- ✓ On the other side **exploitation** means to use the prior knowledge.

Neither the exploration nor the exploitation of prior knowledge can be achieved **without failing in the task**.

Reinforcement Learning

Question:

Reinforcement learning is mostly similar to which natural agent?



Section two: Q-Learning

Machine Learning, 8, 279–292 (1992)

Technical Note

Q-Learning

CHRISTOPHER J.C.H. WATKINS
25b Framfield Road, Highbury, London N5 1UU, England

PETER DAYAN
Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9EH, Scotland

Abstract. Q-learning (Watkins, 1989) is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.

This paper presents and proves in detail a convergence theorem for Q-learning based on that outlined in Watkins

Definition:

- Q-learning is a reinforcement learning **technique**.
- It **learns** a specific **action/value function**, following a specific policy for performing different movements in different situations.

Q-Learning

- ✓ Q-learning model consists of an agent, **states (S)**, and a set of **actions (A)** for each situation.
- ✓ Agent's goal is to **maximize the total reward**.
- ✓ The algorithm has a **function** that **computes** the **state/action** combination.
- ✓ Q-learning has **off-policy** control policy.

Q-learning Strengths:

- ✓ **Ability to learn** with the function **without having a specific model** of the environment.

Q-Learning

Note:

- ✓ Q-learning returns a value specified **by the designer**
- ✓ Then at **each state the agent is rewarded** by a values that **computed** for each **state/action combination**.

Actions?

- ✓ Run left
- ✓ Run right
- ✓ Stop
- ✓ Jump
  - ✓ S/RL/RR
- ✓ Fire

State?

Reward?

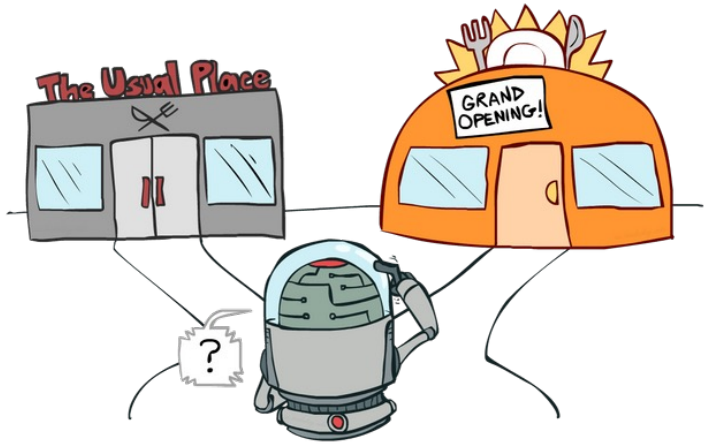
Mario

Q-Learning

Reminder

a) Generate a random number (*rand* $\in [0, 1]$)
b) If *rand* $< \varepsilon$ choose a greedy action from Q-values (**exploitation)**
c) Otherwise choose a random action (**exploration)**

Note: To avoid inefficient exploration, we decrease the parameter ε in time (ε decay)

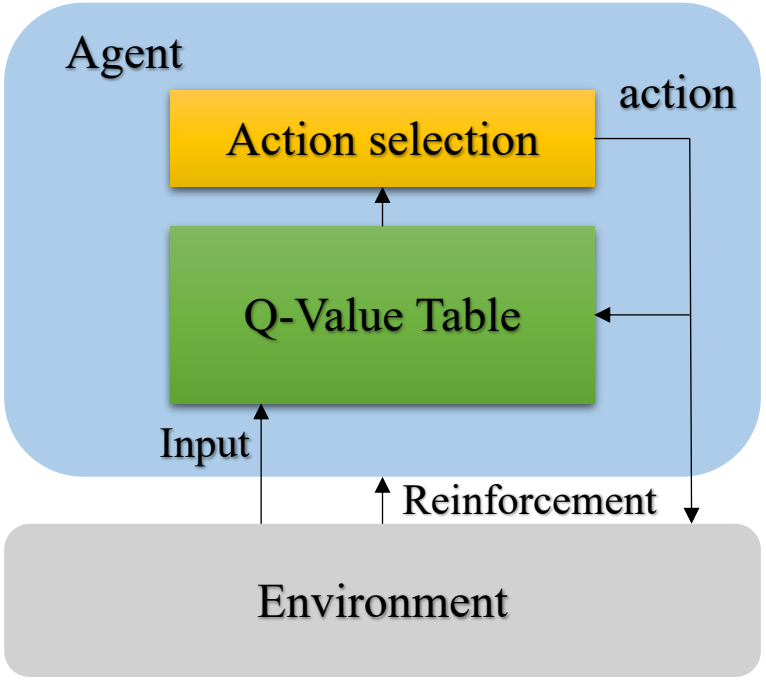


Q-Learning

- ✓ In the environment, **the goal is an state** and **when the agent reaches that state**, the **agent's movement is stops**, (so, the learning process is episodic).

- ✓ In each **episode** the agent is **placed in a random location** and continues to **change Q-values** until it reaches the target state, when the episode ends and the new episode begins.

So what was the **episode**?



Q-Learning

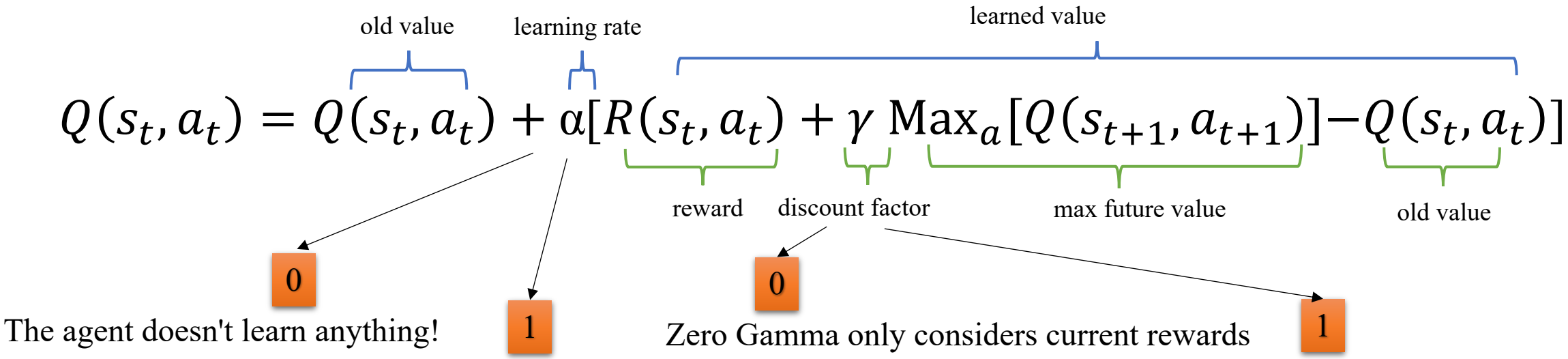
- ✓ In the simplest form in Q-learning, tables are used to store data.
- ✓ A example of a reward table:

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

- ✓ The values in this table can be filled with a random initial value (or zero).
- ✓ The agent periodically detects the current state of (**s**) and performs an action (**a**).
- ✓ It then observes the reward of **r (s, a)** as well as the new state of action:

Q-Learning

- ✓ The core of the algorithm consists of a loop of update of the Tables (we will see).
- ✓ Therefore, it is correcting its knowledge based on new information.



Learning rate:

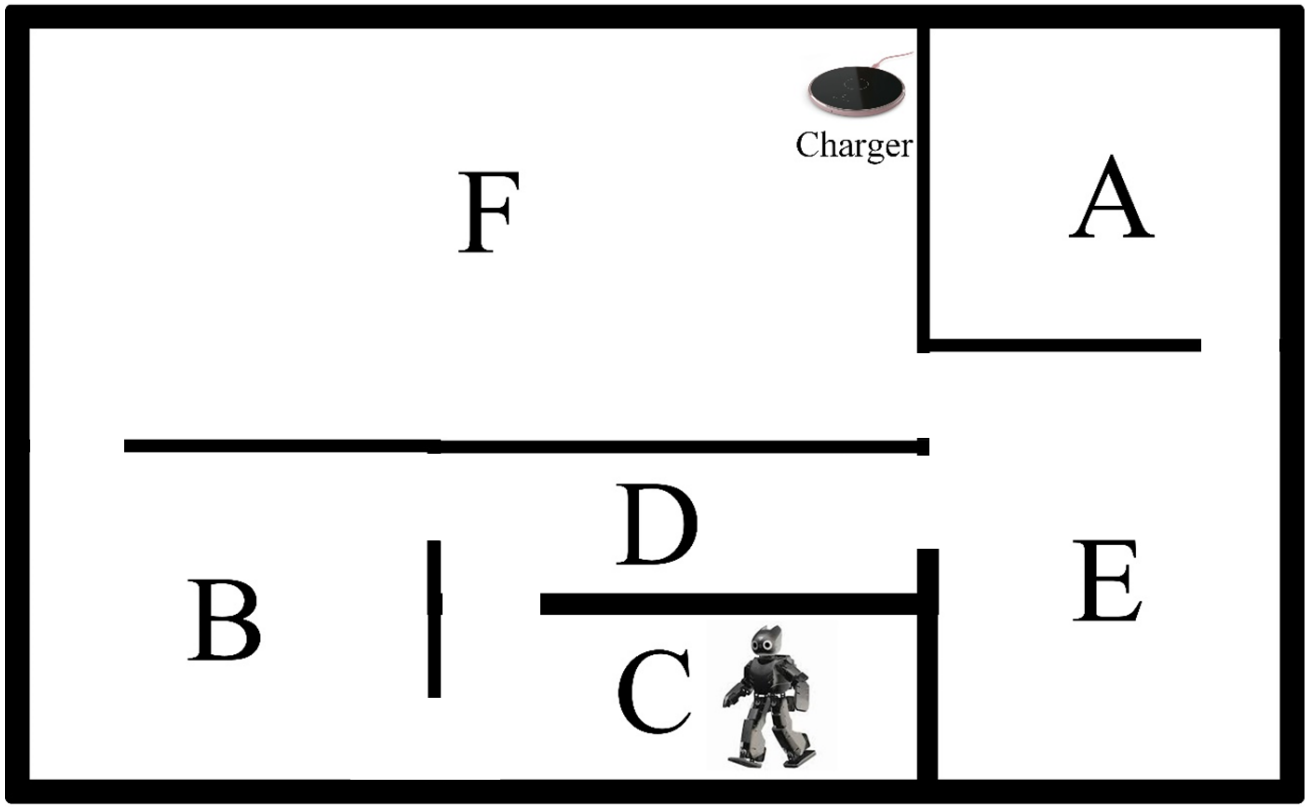
- ✓ How much new information will be preferred over old information.

Discount factor:

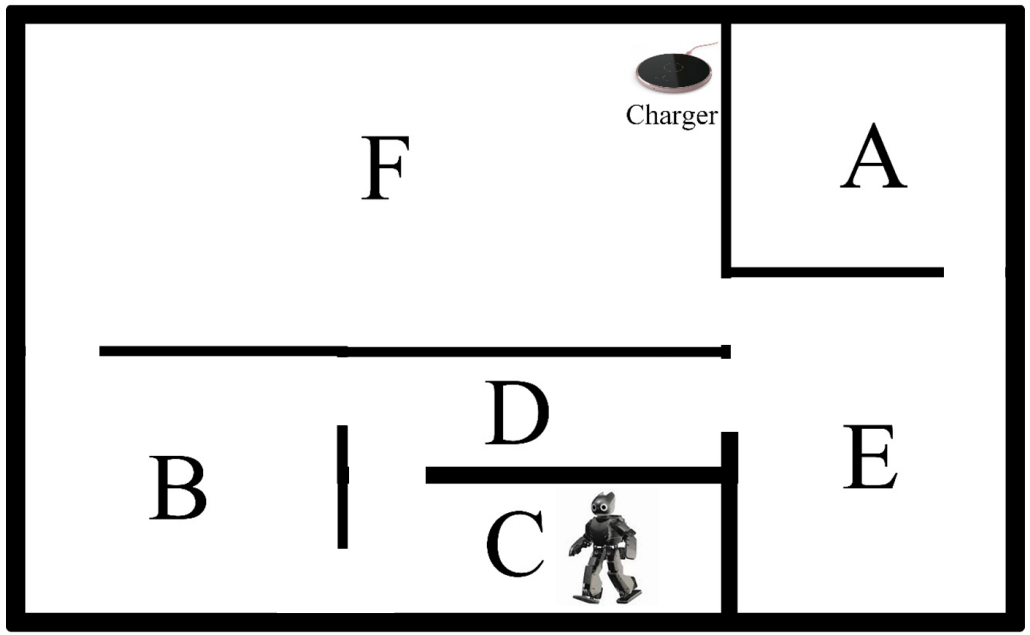
- ✓ The discount factor determines the importance of future rewards.

Q-Learning

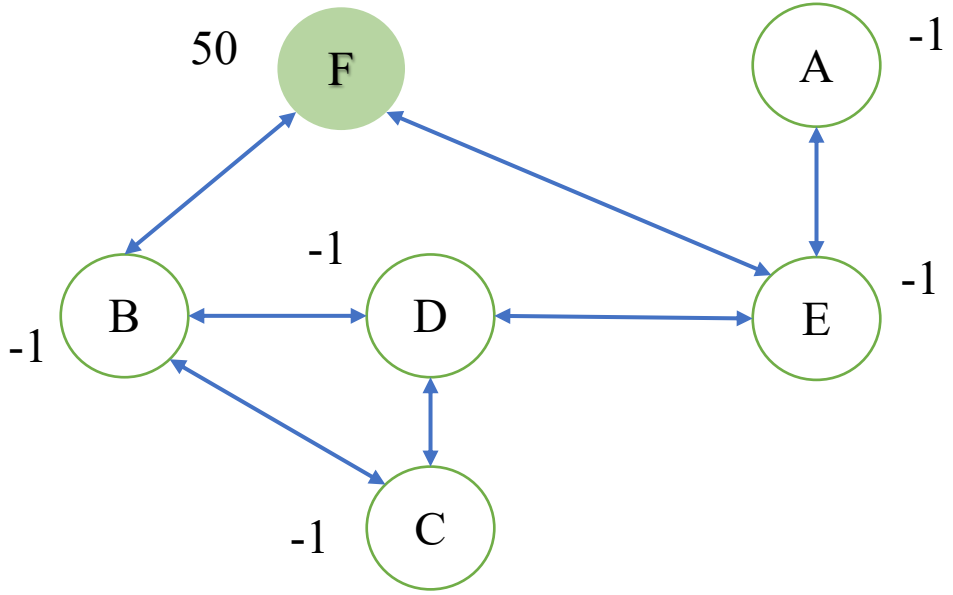
Example of an environment:



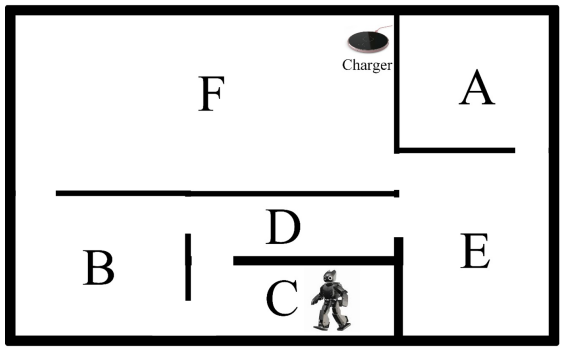
Q-Learning

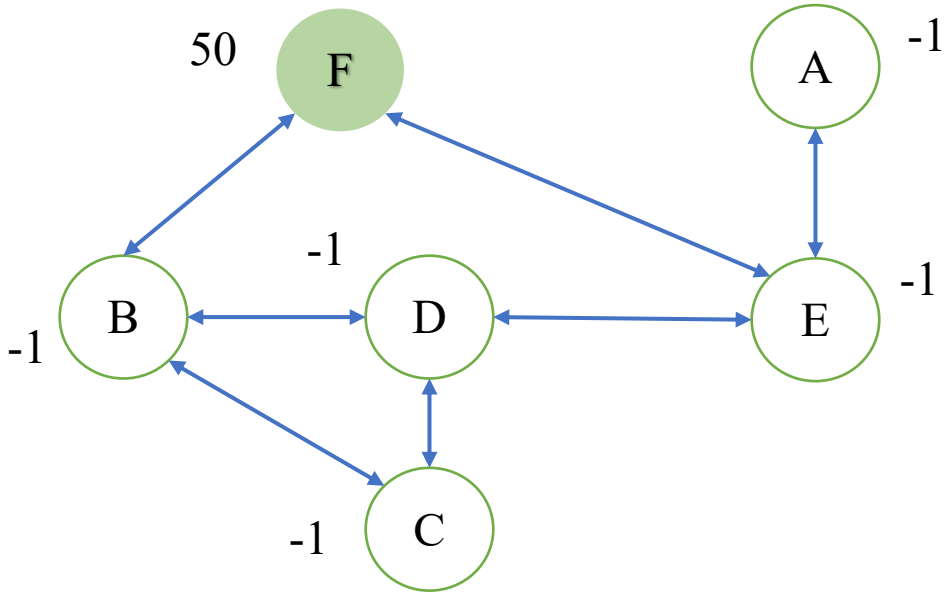


MDP



Q-Learning





Reward Table

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

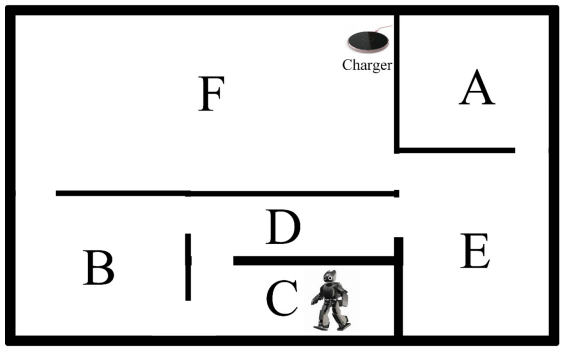
Q-Table

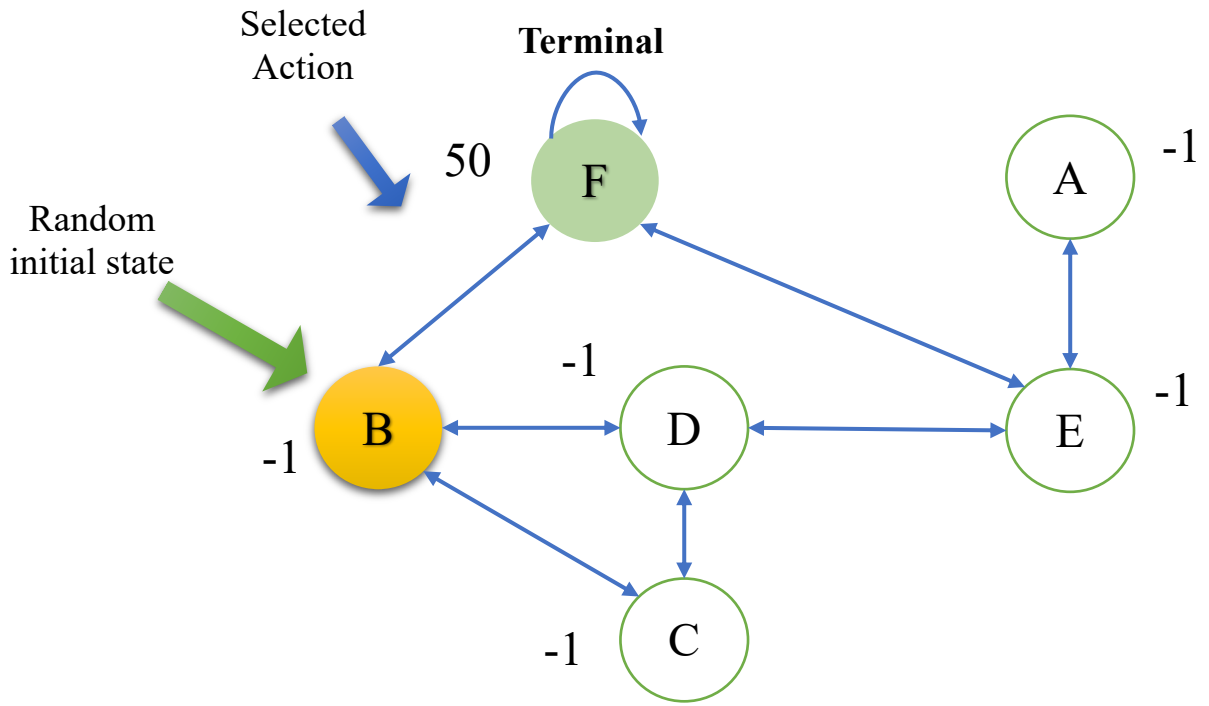
$$Q = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Important key point:

- Action is considered as a series of movements of agent like walking between two rooms here right.
- So what is the action in a 6 DOF manipulator then?
- Be careful about state space.

Q-Learning





Reward Table

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

Update

Q-Table

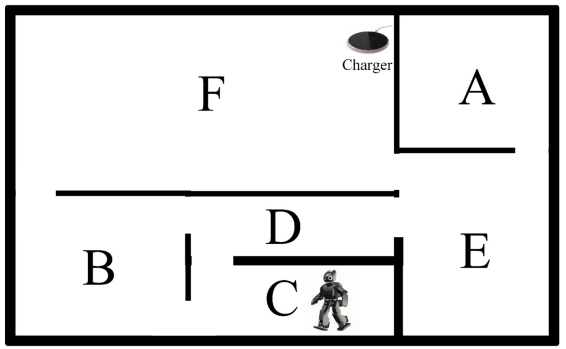
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

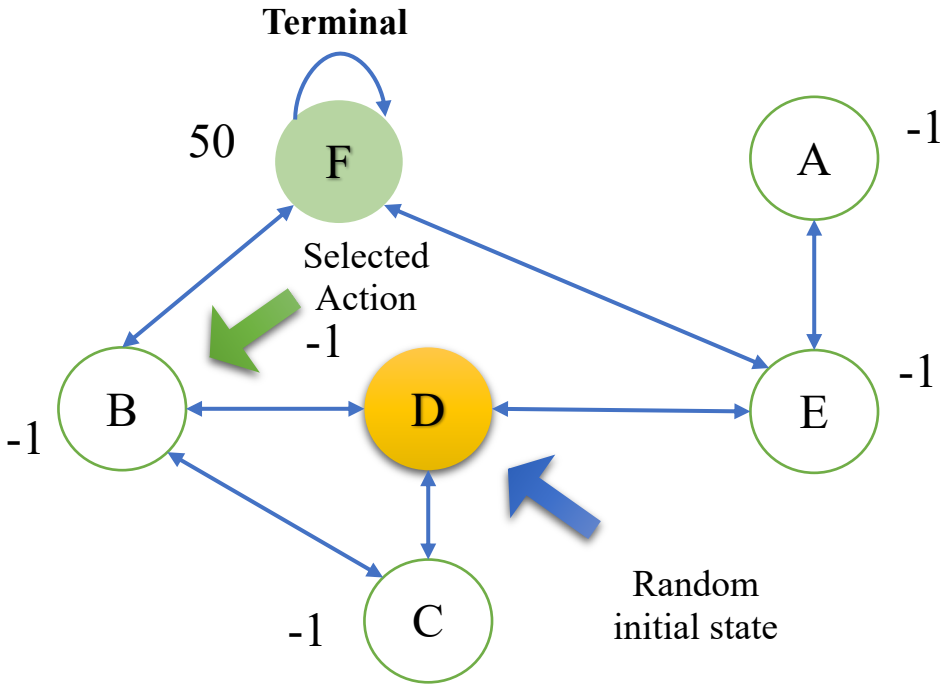
$$Q(B,F) = Q(B,F) + 0.9 * [R(B,F) + 0.8 * Max_a[Q(F,B), Q(F,E), Q(F,F)] - Q(B,F)]$$

$$= 0 + 0.9 * [50 + 0.8 * \mathrm{Max}[0, 0, 0] - 0] = 45$$

$$Q = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 45 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning





Reward Table

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

Update

Q-Table

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

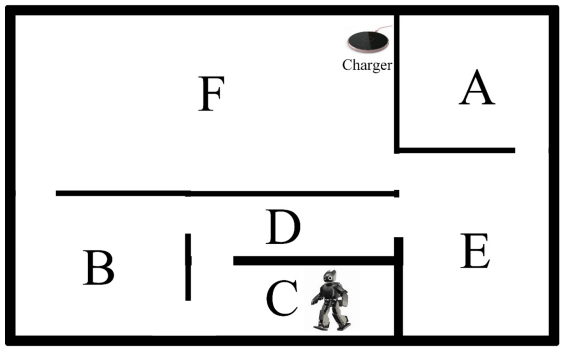
$$Q(D,B) = Q(D,B) + 0.9 * [R(D,B) + 0.8 * Max_a[Q(B,F), Q(B,D), Q(B,C)] - Q(D,B)]$$

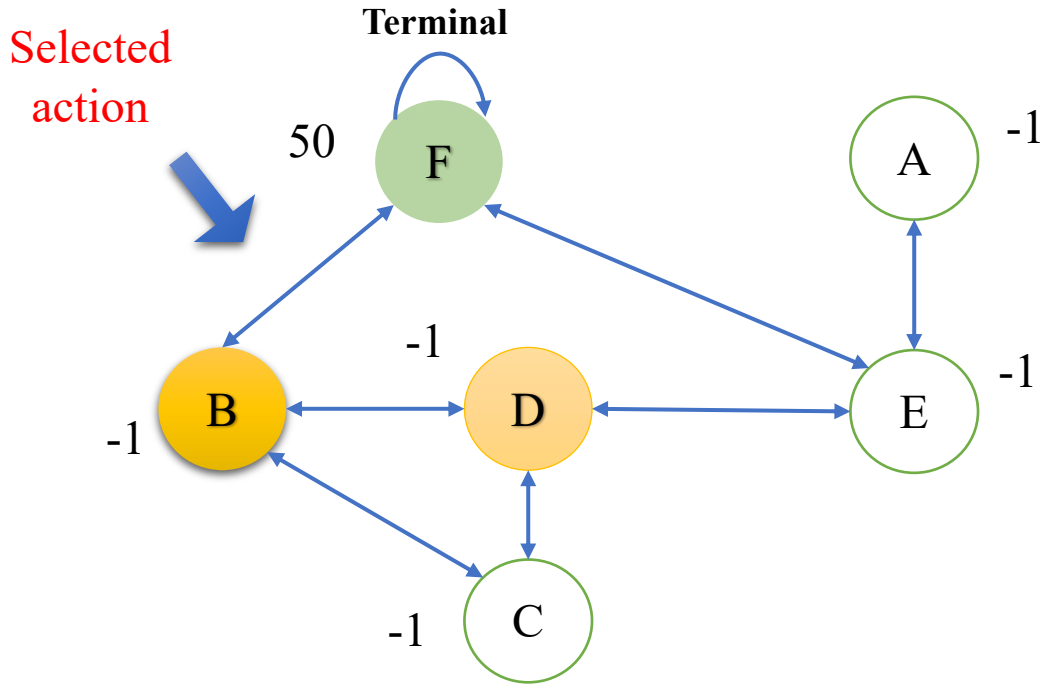
$$= 0 + 0.9 * [-1 + 0.8 * \mathrm{Max}[45, 0, 0] - 0] = 31.5$$

$$Q = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 45 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 31.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The next state, B, becomes the current state; and, since it is not the target state, the inner loop of algorithm Q-value will be repeated.

Q-Learning





Reward Table

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

Update

Q-Table

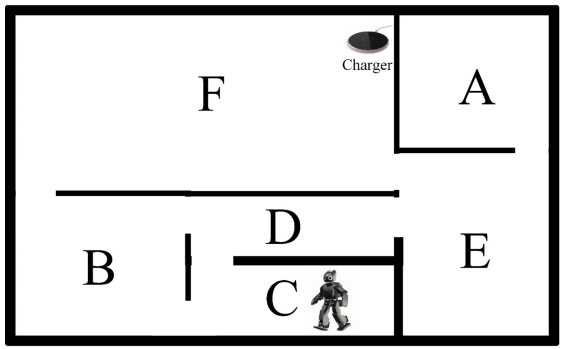
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma\ \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

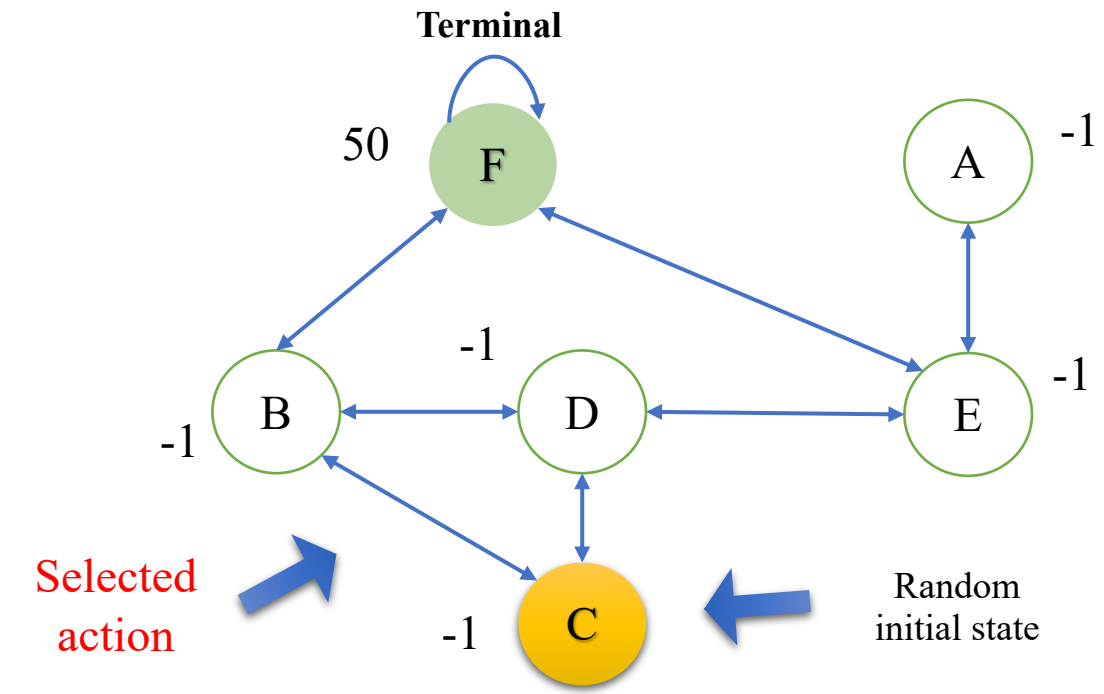
$$Q(B,F) = Q(B,F) + 0.9 * [R(B,F) + 0.8 * Max_a[Q(F,B), Q(F,E), Q(F,F)] - Q(B,F)]$$

$$= 45 + 0.9 * [50 + 0.8 * \mathrm{Max}[0, 0, 0] - 45] = 49.5$$

$$Q = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 49.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 31.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning





Reward Table

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

Update

Q-Table

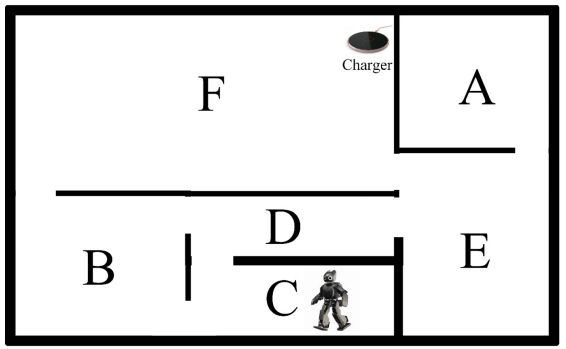
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma\ \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

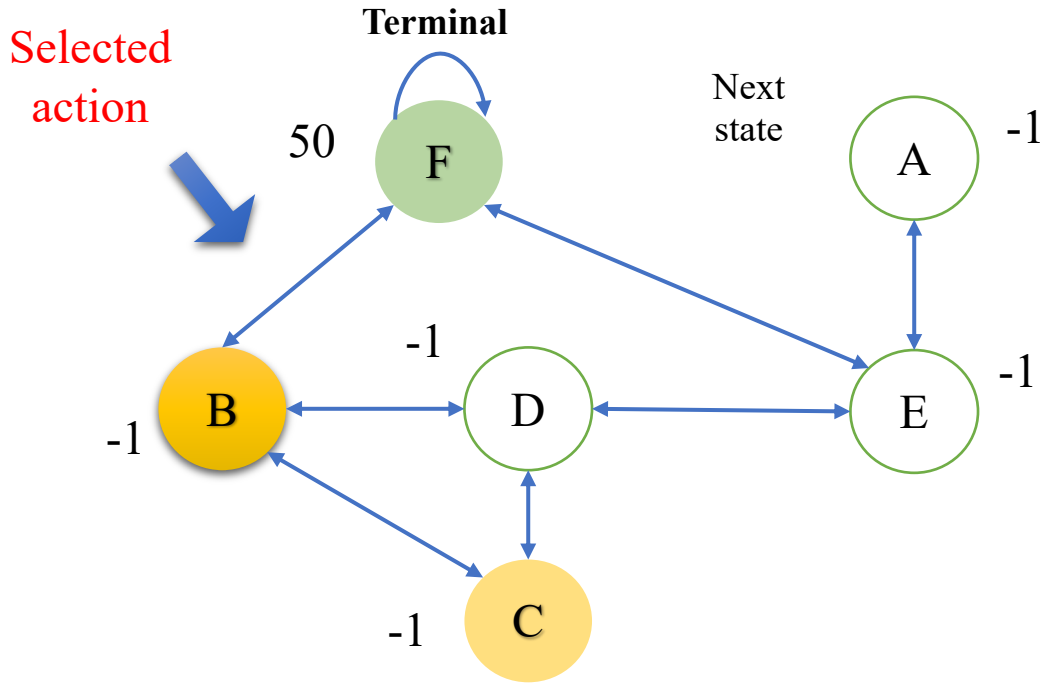
$$Q(C,B) = Q(C,B) + 0.9 * [R(C,B) + 0.8 * Max_a[Q(B,D), Q(B,F)] - Q(C,B)]$$

$$= 0 + 0.9 * [-1 + 0.8 * \mathrm{Max}[0, 0] - 0] = -0.9$$

$$Q = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 49.5 \\ 0 & -0.9 & 0 & 0 & 0 & 0 \\ 0 & 35.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning





Reward Table

$$R = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & -1 & - \\ - & - & - & -1 & - & 50 \\ - & - & - & -1 & - & - \\ - & -1 & -1 & - & -1 & - \\ -1 & - & - & -1 & - & 50 \\ - & -1 & - & - & -1 & 50 \end{bmatrix} \end{array}$$

Update

Q-Table

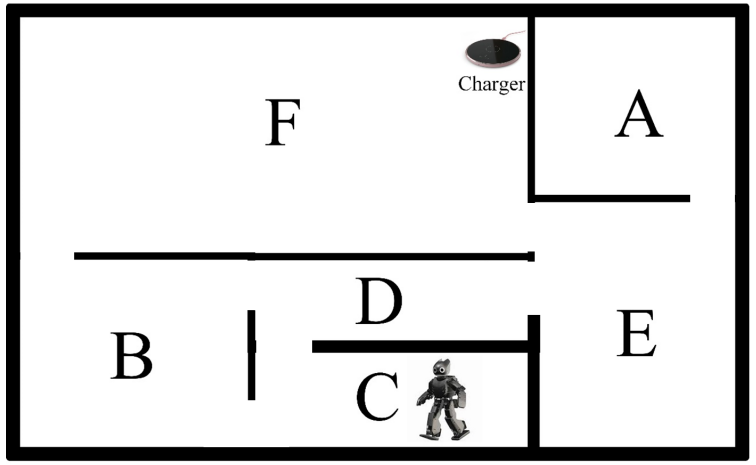
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma\ \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

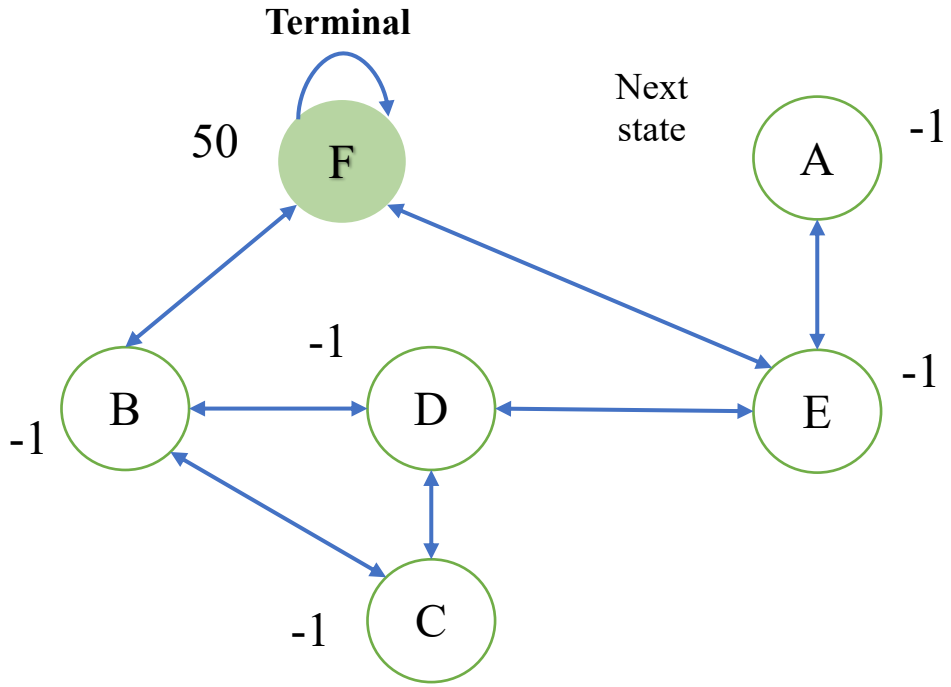
$$Q(B,F) = Q(B,F) + 0.9 * [R(B,F) + 0.8 * Max_a[Q(F,B), Q(F,E), Q(F,F)] - Q(B,F)]$$

$$= 49.5 + 0.9 * [50 + 0.8 * \mathrm{Max}[0, 0, 0] - 49.5] = 49.95$$

$$Q = \begin{array}{c} (s,a) \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 49.95 \\ 0 & -0.9 & 0 & 0 & 0 & 0 \\ 0 & 35.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning



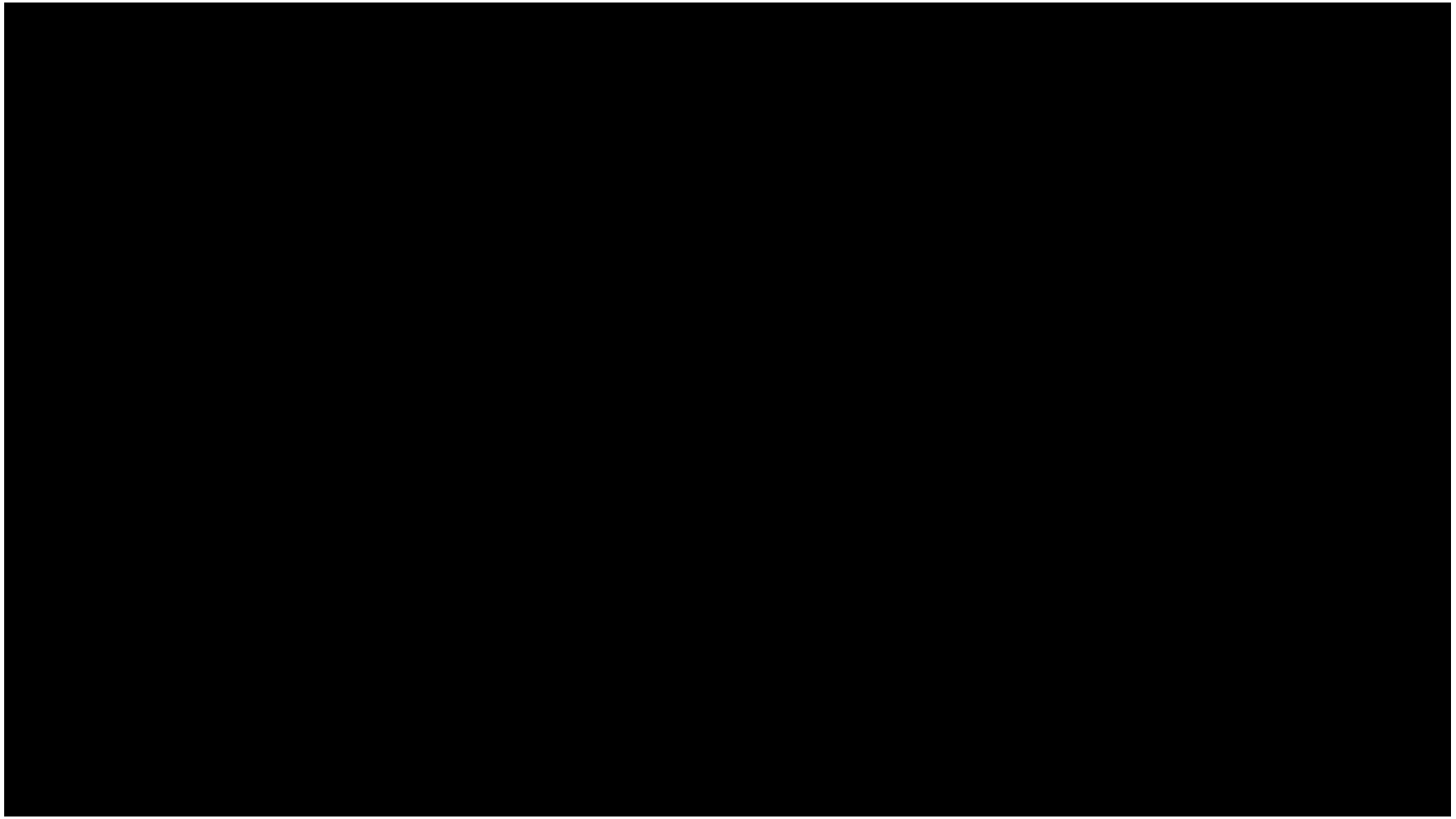


Assignment: Implement the discussed environment in python:

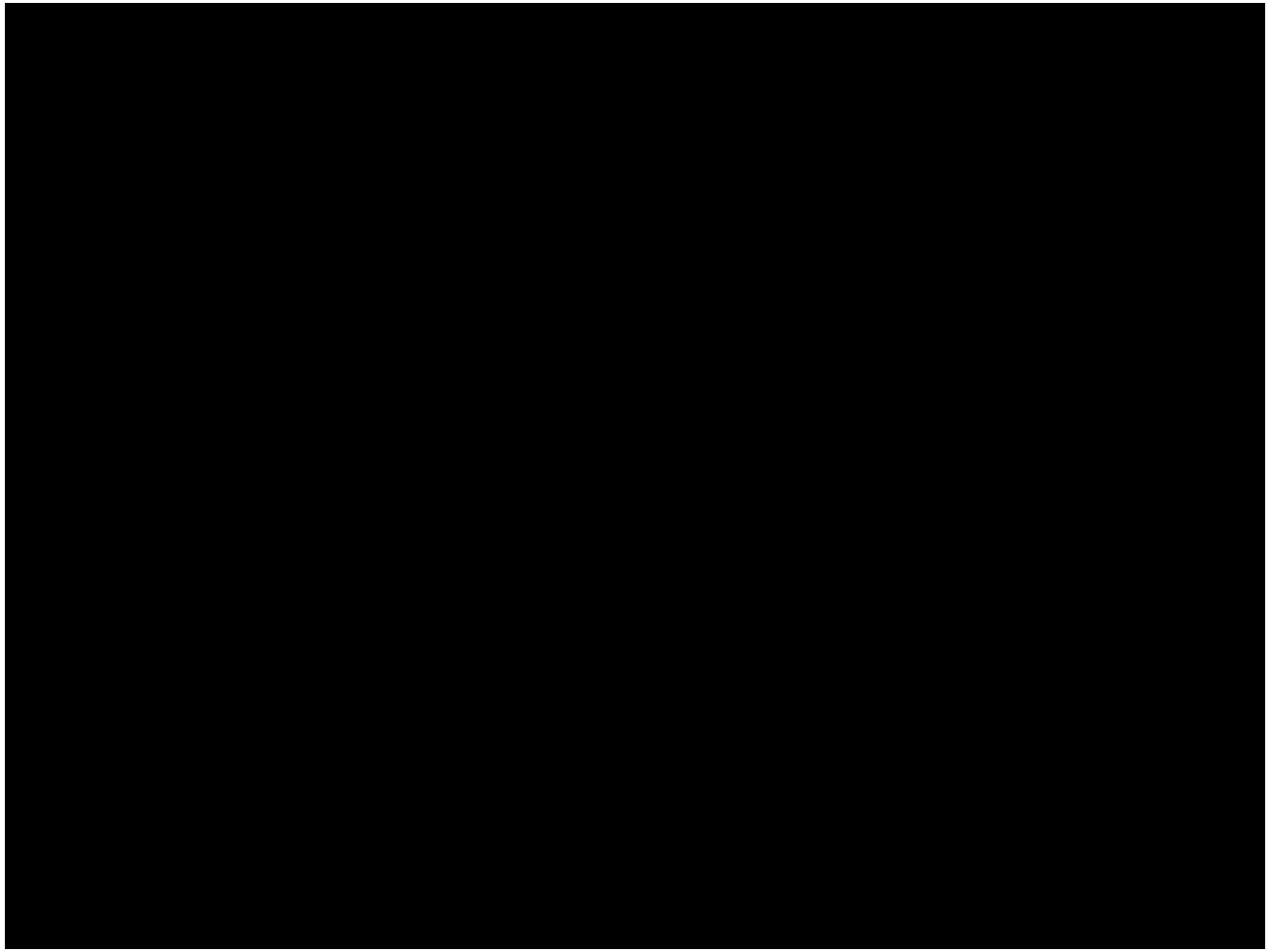
1) Report the Q-Table values convergence plot (chose number of episodes to show convergence).

2) Change the initial state as A and target state to C and report the Q-Table and convergence plot.

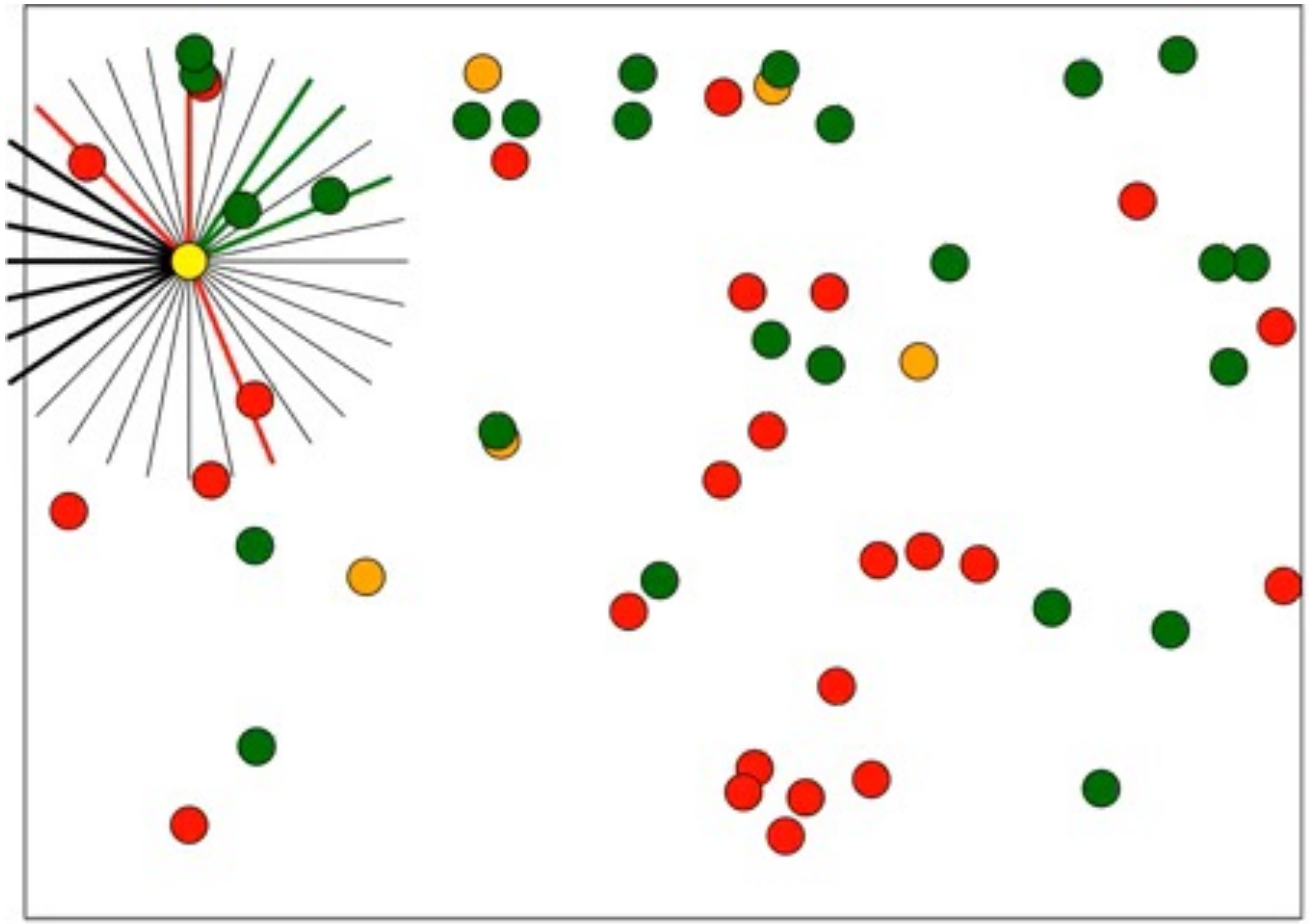
Q-Learning: Simple Examples



Q-Learning: Simple Examples



Q-Learning: Simple Examples



SARSA

SARSA (State-Action-Reward-State-Action)

- ✓ Similar to Q-learning algorithm.
  - ➢ Q-learning and SARSA are **both policy control methods** which work on evaluating the optimal Q-value for all action-state pairs.

Differences:

- ✓ SARSA is an **on-policy** TD control method.
  - ➢ So, it doesn't follow a policy to find the next action but **chooses the action in a greedy** fashion (certain policy).

SARSA

Differences:

Q-learning $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$

old value; learning rate; learned value; reward; discount factor; max future value; old value

SARSA $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

Selected next action e.g. after applying ε-greedy

Expected SARSA

✓ Is similar to SARSA But takes the expectation (mean) of Q-values for every possible action.

SARSA $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

E-SARSA $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \sum \pi(a|s_{t+1}) Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

weighted sum of all possible next actions

Note: weighs are probability of taking each action in agent policy

✓ Updating Q value has **lower variance** which seems to be better than SARSA but it is **more expensive.**

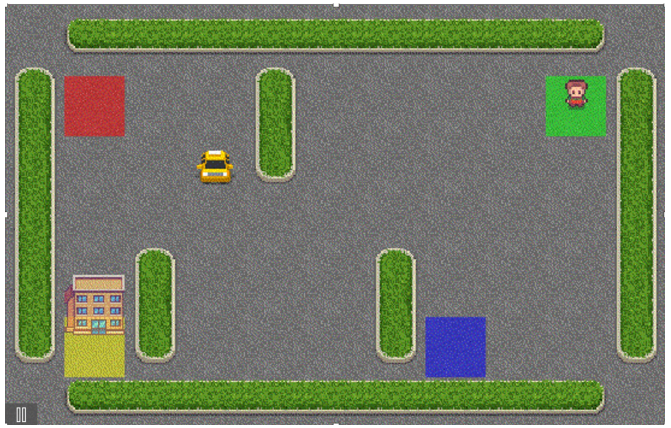
Comparison

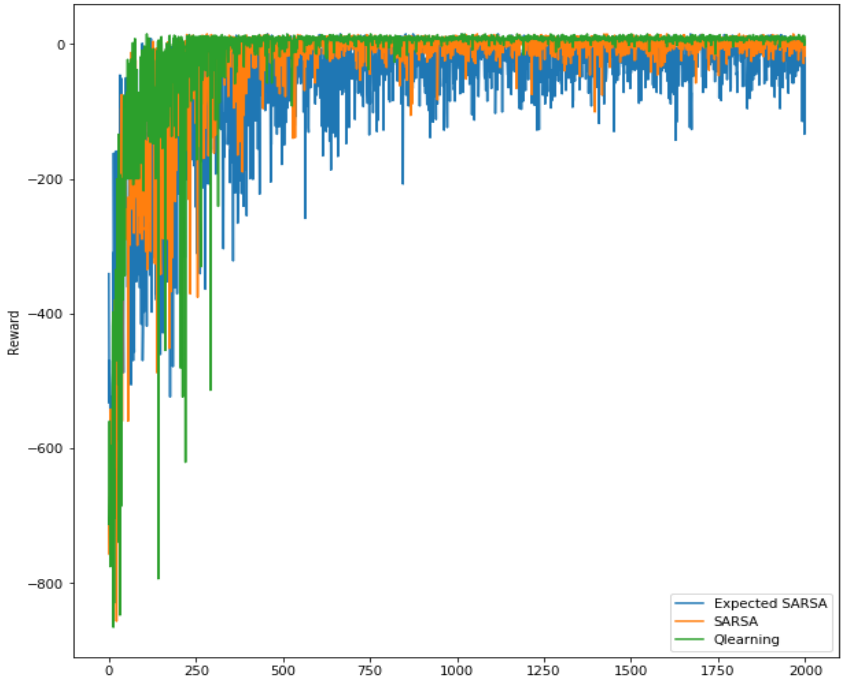
OpenAI gym:

- ✓ A toolkit for developing and comparing reinforcement learning algorithms.
- ✓ Easy-to-use
- ✓ Open source interface to reinforcement learning tasks

Environment:

- ✓ Step
- ✓ Render
- ✓ Reset
- ✓ Close





https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e

Clone gym-examples:

```
git clone https://github.com/Farama-Foundation/gym-examples
cd gym-examples
python -m venv .env
source .env/bin/activate
pip install -e .
```

https://www.gymlibrary.dev/

Policy

On-Policy vs Off-Policy

- ✓ On-policy and off-policy learning methods is only related to the **evaluating of Q-value** $Q(s, a)$.

On-Policy

Methods to **learn the $Q(s, a)$ function** by the **actions** that we used from current policy $\pi(a|s)$.

Off-Policy

Methods to **learn the $Q(s, a)$ function** by different actions than our current policy (e.g. can be random)

Policy

On-Policy vs Off-Policy

Q-learning

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

learning rate; learned value; old value; reward; discount factor; max future value; old value

Important: Q-learning is **off-policy** because max may choose something else that the policy that what it could be in ε-greedy

SARSA

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Selected next action on-policy

It means that after **applying ε-greedy based action agent moves to next state s_{t+1} and then use its Q-value** for update the current states Q-vale. **(reason for being on-policy)**

Policy

On-Policy vs Off-Policy

✓ **On-policy SARSA** algorithm:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

➢ Where a_{t+1} is the action, that was **taken according to policy π**.

✓ **Off-policy Q-learning** algorithm:

updated policy is different from the behavior policy

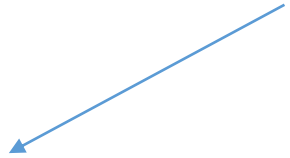
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma\, \mathrm{Max}_a[Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)]$$

➢ Where a_{t+1} are all actions, that were explored in state s_{t+1}

Policy

On-Policy vs Off-Policy

E-SARSA $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \sum \pi(a|s_{t+1}) Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$



- ✓ If for all actions:
  - ➢ **We have** on-policy E-SARSA
- ✓ If for some of actions or random:
  - ➢ **We have** off-policy E-SARSA

Why to use Off-Policy?

Some benefits of Off-Policy methods:

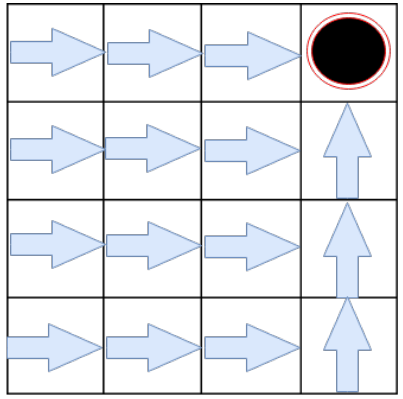
- ✓ **Continuous exploration:**
  - ➢ As an agent is learning other policy then it can be used for **continuing exploration while learning optimal policy**.
- ✓ **Learning from Demonstration:** Agent can learn from the demonstration.
- ✓ **Parallel Learning:** This speeds up the convergence i.e learning can be fast.

Why? Next slide

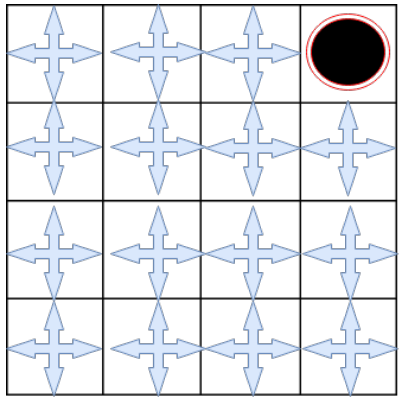
Policy

1. **Target Policy $\pi(a|s)$:** It is the policy that an agent is trying to learn i.e agent is learning value function for this policy.

2. **Behavior Policy b(a|s):** It is the policy that is being used by an agent for action select i.e agent follows this policy to interact with the environment.

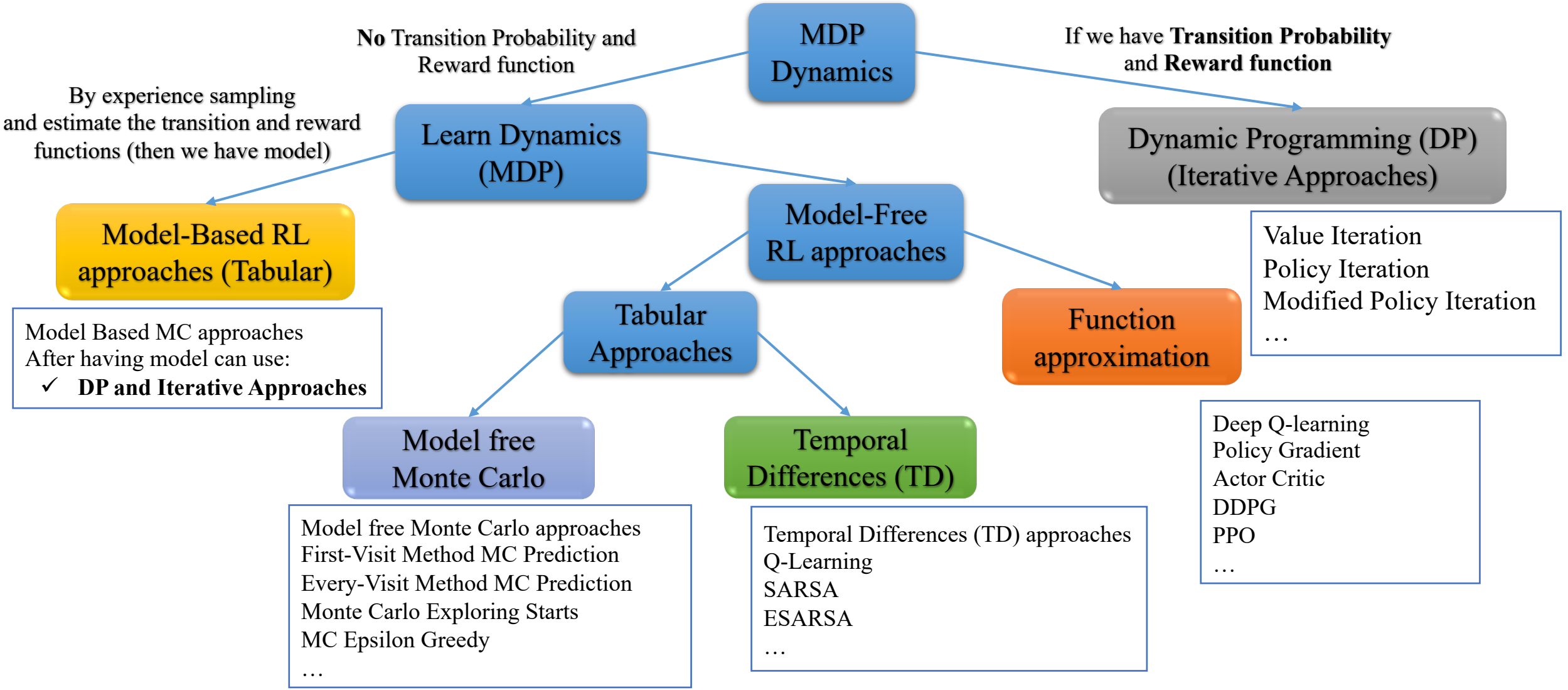


Target Policy



Behavior Policy

Reinforcement Leering in General Perspective



Section Three: Path Navigation For Indoor Robot With Q-Learning

INTELLIGENT AUTOMATION AND SOFT COMPUTING, 2016
http://dx.doi.org/10.1080/10798587.2015.1095485

Path Navigation For Indoor Robot With *Q*-Learning

One simple paper with Q-learning

Lvwen Huang[a], Dongjian He[b], Zhiyong Zhang[a] and Peng Zhang[c]

[a]College of Information Engineering, Northwest A&F University, Yangling, China; [b]College of Mechanical and Electronic Engineering, Northwest A&F University, Yangling, China; [c]School of Economics and Management, Xi'an University of Technology, Xian, China

ABSTRACT

A *Q*-learning based path navigation method is proposed and validated in this paper for solving the moving control along specified path of real indoor mobile robot. A grid and topological indoor corridor environment map is employed and characterized by a set of geometrical scale invariant key-points. The navigation strategy is composed of on-line and off-line stages with the same components redefinitions or definitions of *Q*-learning. During the off-line learning stage, the personal computer records the optimal path after computer learning simulation, and then the path is sent to the robot through wireless data radio with RS232. At the on-line navigation stage, the robot calculates the relative positions between the locations along this optimal path, and then navigates the environment autonomously. The experiments on computer simulation and an actual robot have been verified the superior effectiveness and applicability of the proposed strategy.

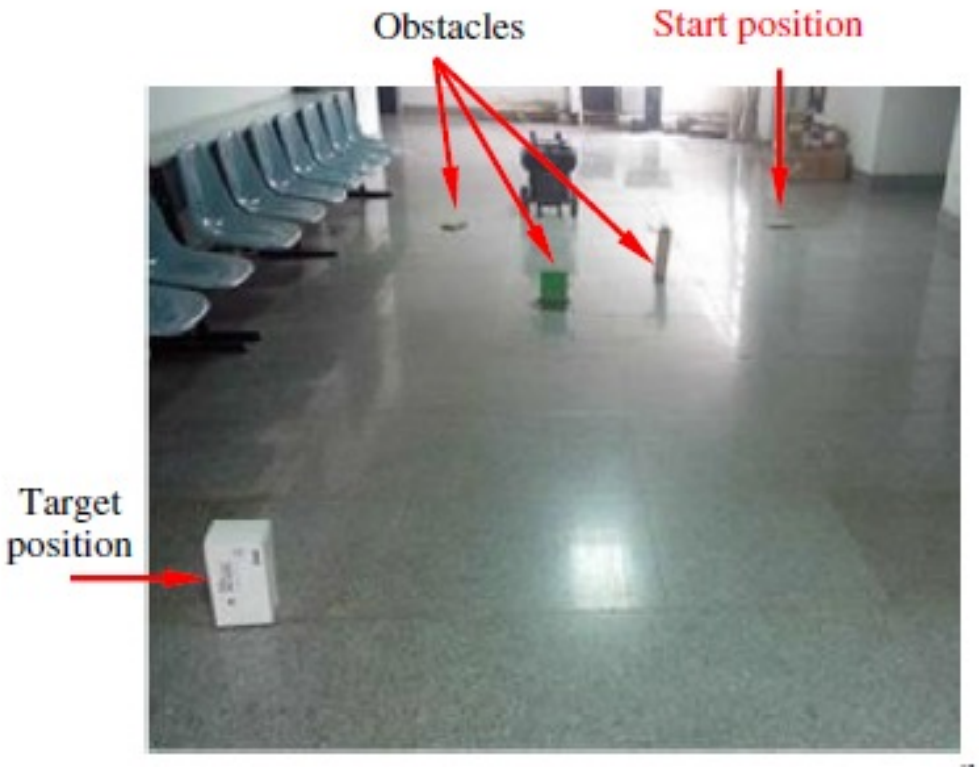
KEYWORDS

Navigation; indoor robot; Q-learning; map

ary 2016

Path Navigation For Indoor Robot With Q-Learning

Target:
A robot that can learn the shortest path to the target in the real world.

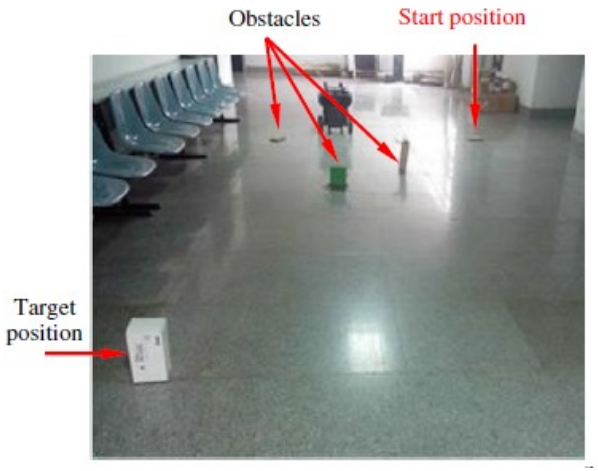


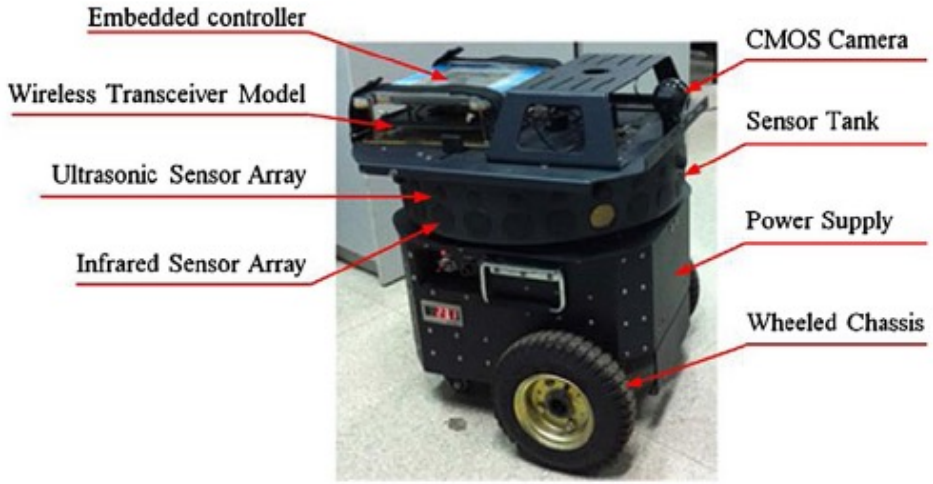
Path Navigation For Indoor Robot With Q-Learning

solution:

- A Q-Learning model based on robot motion is presented.
- The robot's strategy of moving is based on two stages, **online** and **offline**.

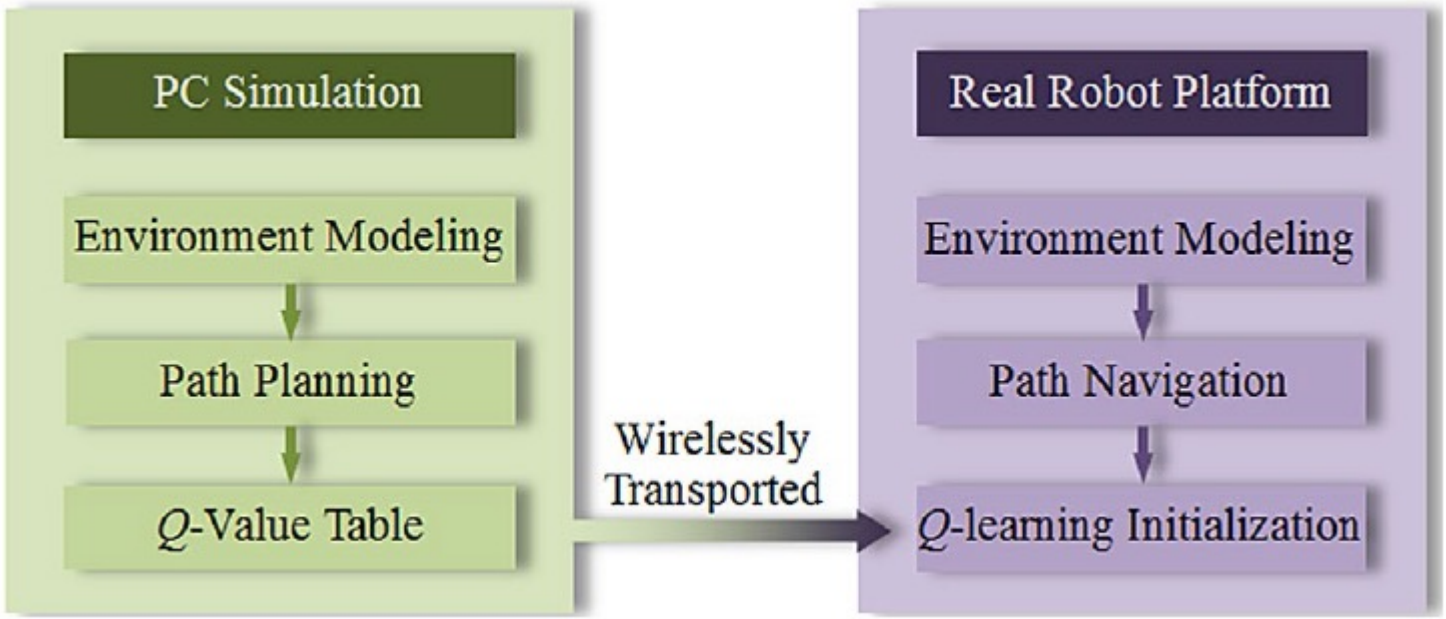
  Attributes:
  - ✓ The robot only knows its starting point and current location.
  - ✓ The robot knows the target coordinates (it cannot get its distance from the environment with the distance sensors.
  - ✓ The robot has impedance avoidance sensors.
  - ✓ The environment can be dynamic and unpredictable.





Path Navigation For Indoor Robot With Q-Learning

Robot Motion Architecture



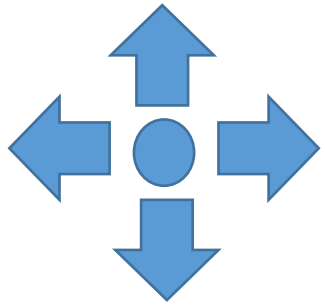
Path Navigation For Indoor Robot With Q-Learning

Used equation in the proposed method:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_a, a_t) + \gamma \max_{a_{t+1} \in A_s} Q(s_{t+1}, a_{t+1}) - Q(a_t, a_t)]$$

Path Navigation For Indoor Robot With Q-Learning

Avoidance Strategies



Reminder: When a robot once reaches the environment and finds a target, we call that one episode.
In this case, after completing an episode, the robot returns to its first initial pre-defined position.

Min Dis to All Obstacles

Min safe Dis to Obstacles

Dis from Robot to target

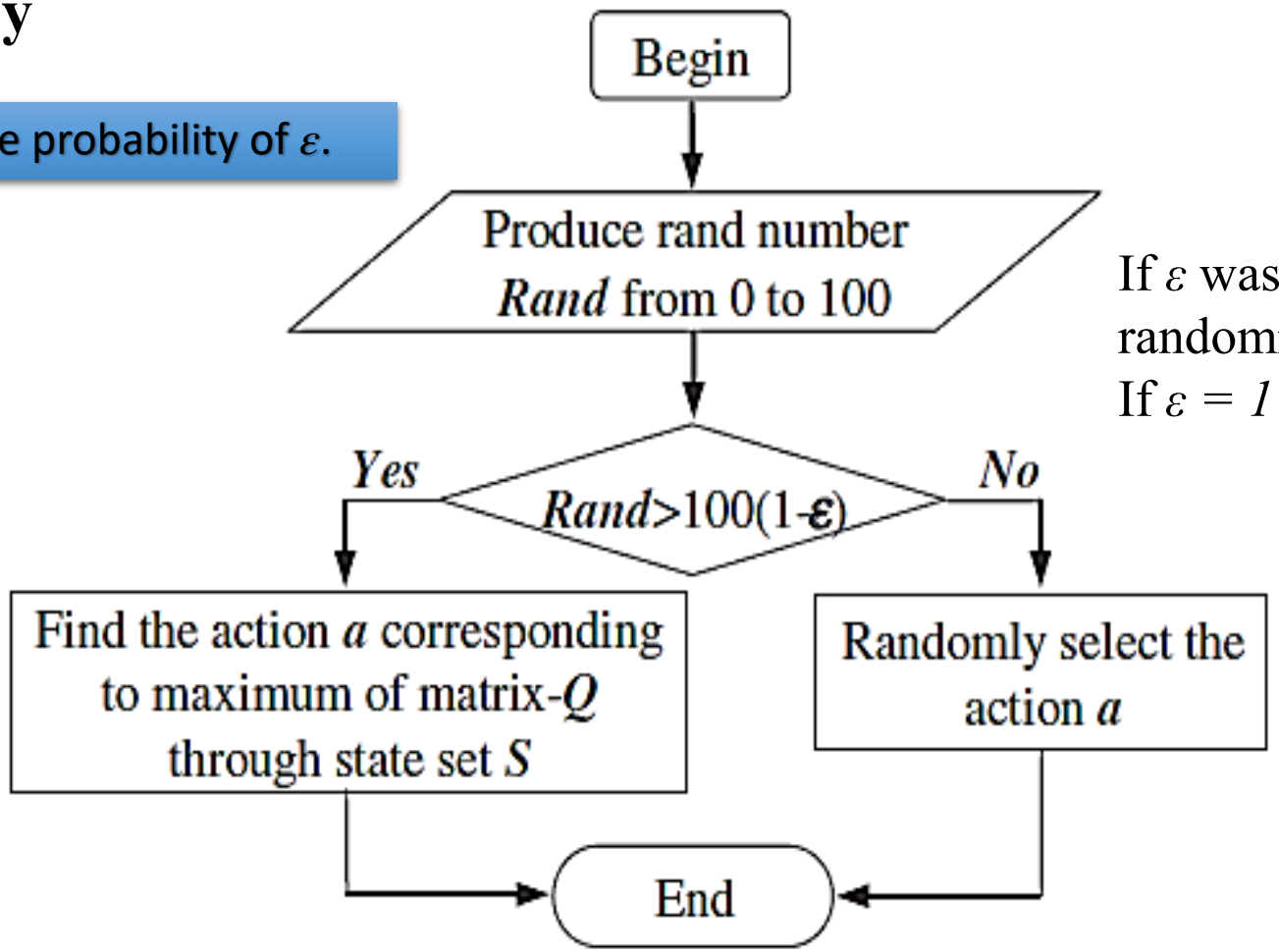
Note: The immediate reward from the environment has a direct impact on the speed and accuracy of the learning.

$$r = \begin{cases} -2 & d_{\min} < d_s, collision \\ -0.2 & d_{\min} = d_s \\ 0 & otherwise \\ 2 & d_o < d_s, \text{i.e.} d_0 = 0 \end{cases}$$

Path Navigation For Indoor Robot With Q-Learning

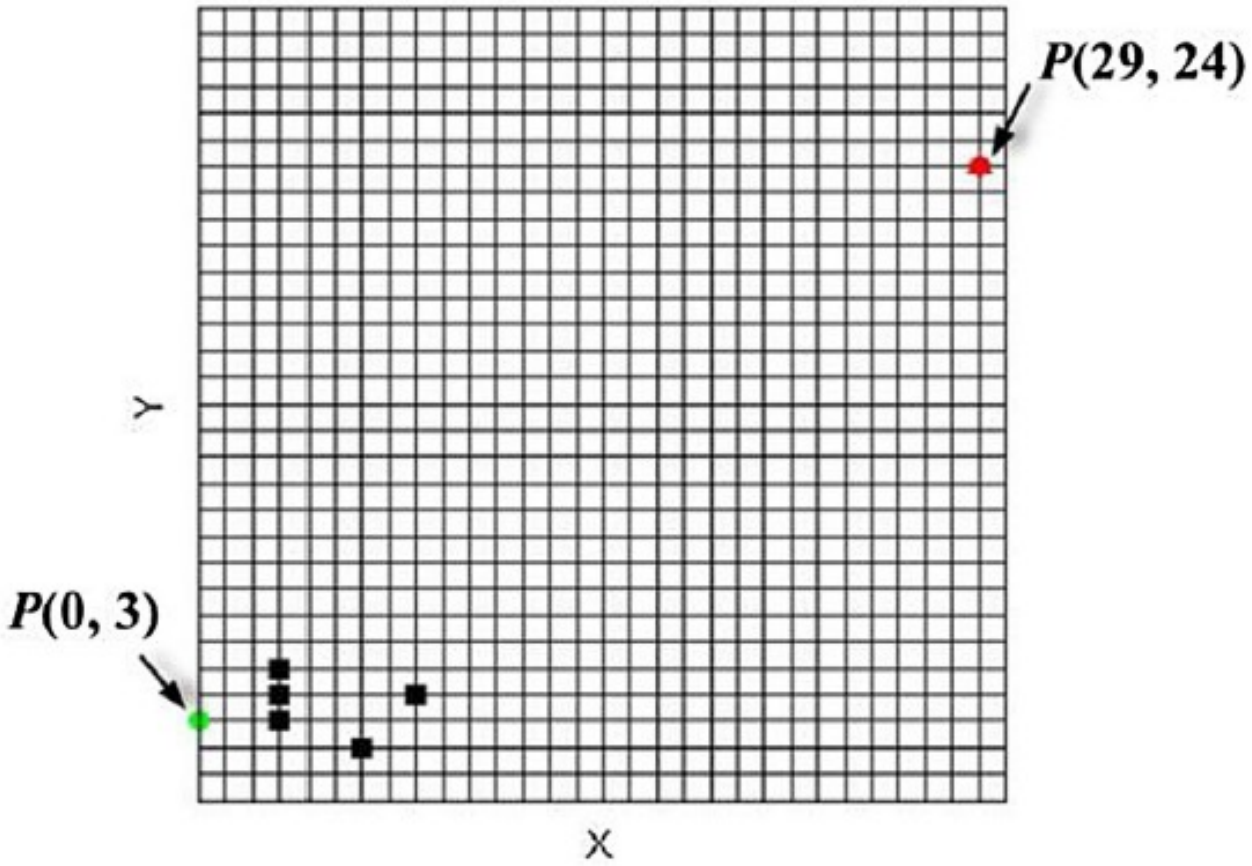
:ε-greedy policy

Behavior with the probability of ε.

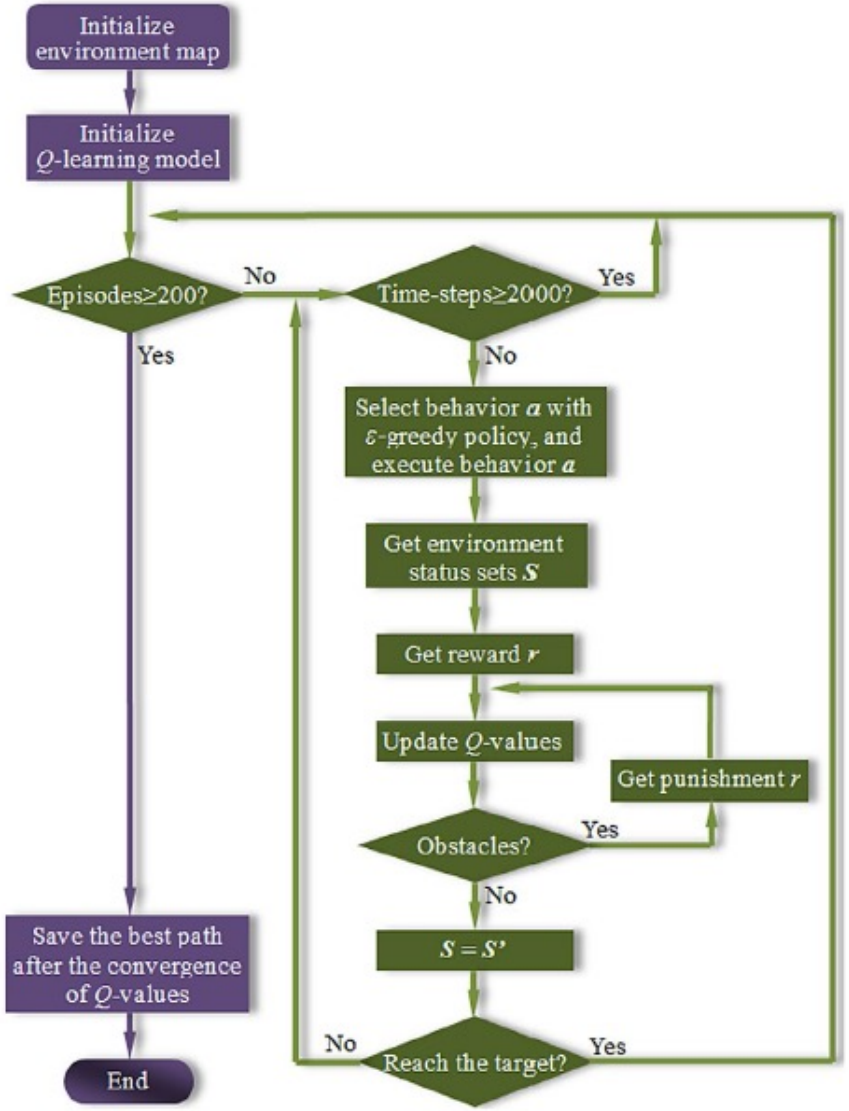


If ε was too small, it could increase the randomness of the action selection.
If $\varepsilon = 1$ *then maximum*

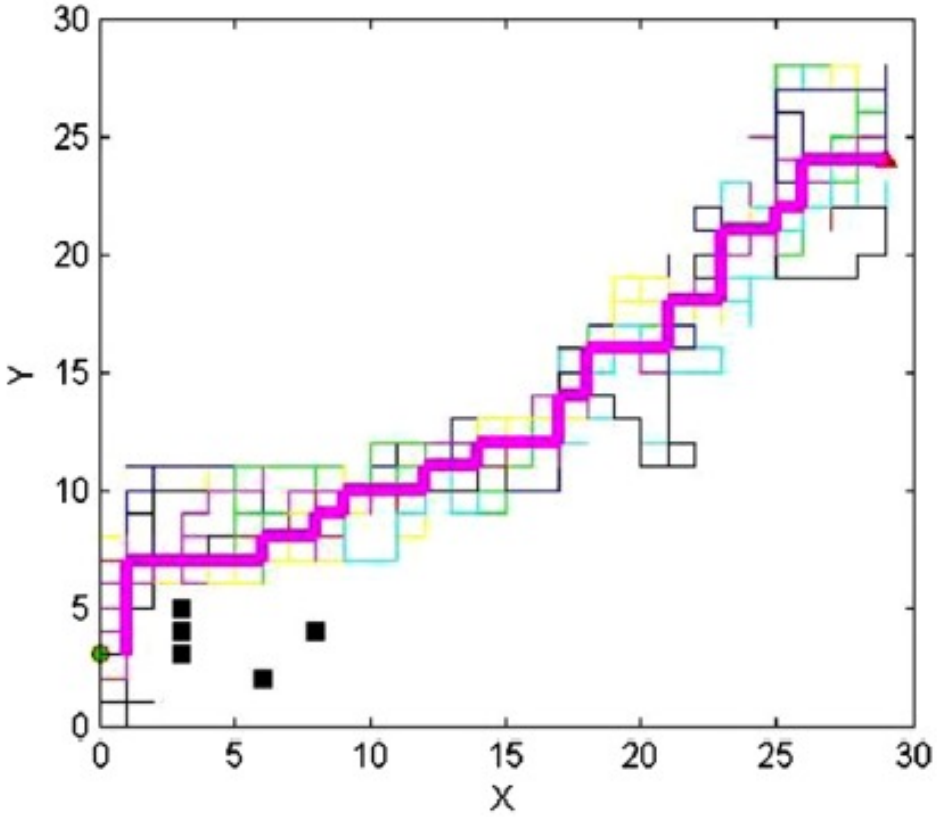
Path Navigation For Indoor Robot With Q-Learning



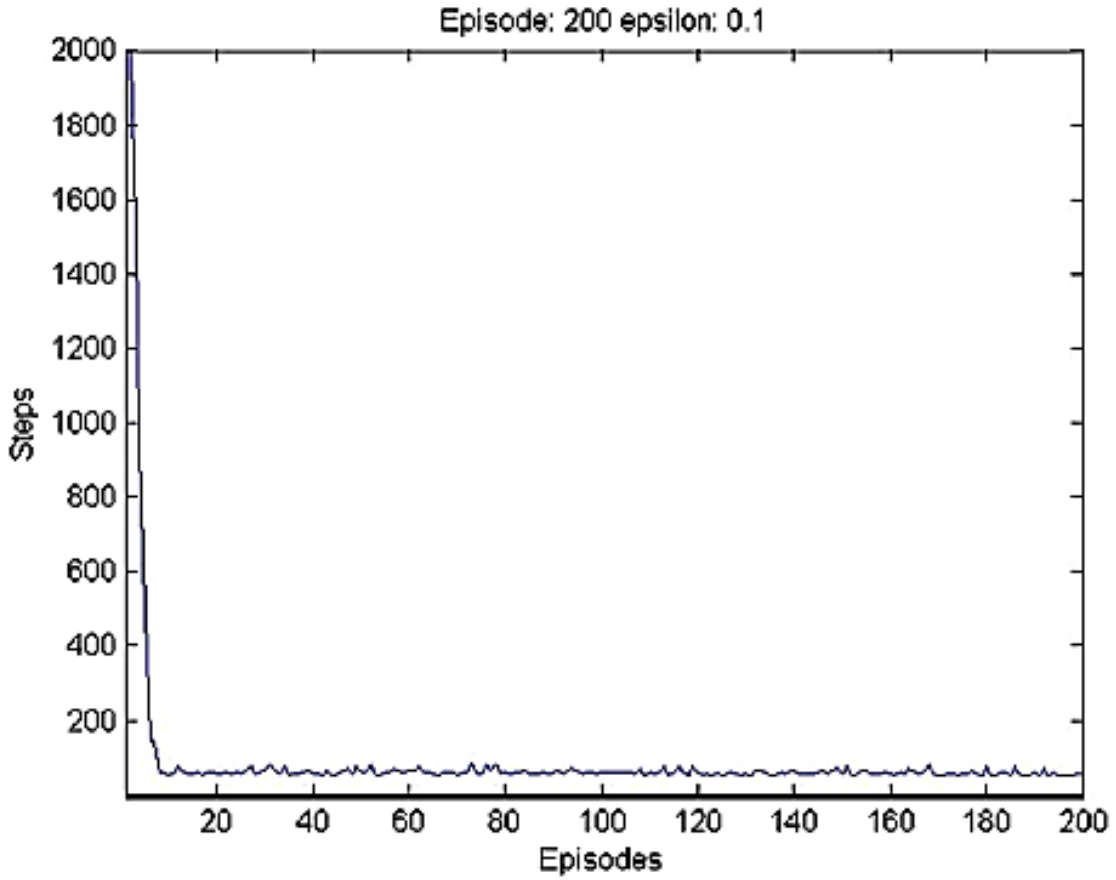
Path Navigation For Indoor Robot With Q-Learning



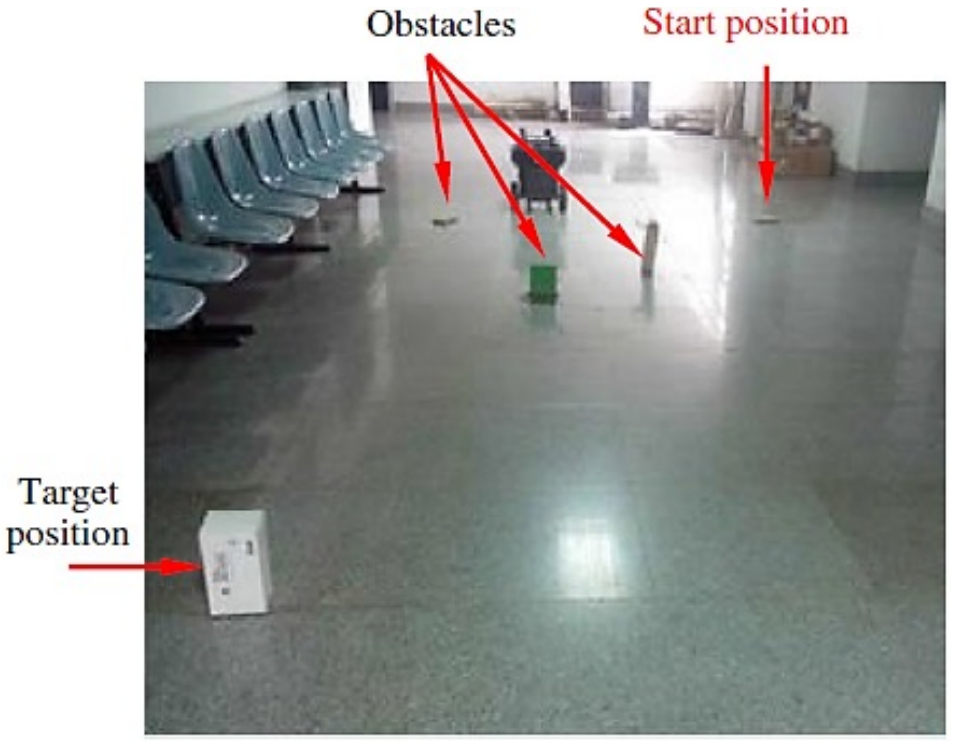
Path Navigation For Indoor Robot With Q-Learning



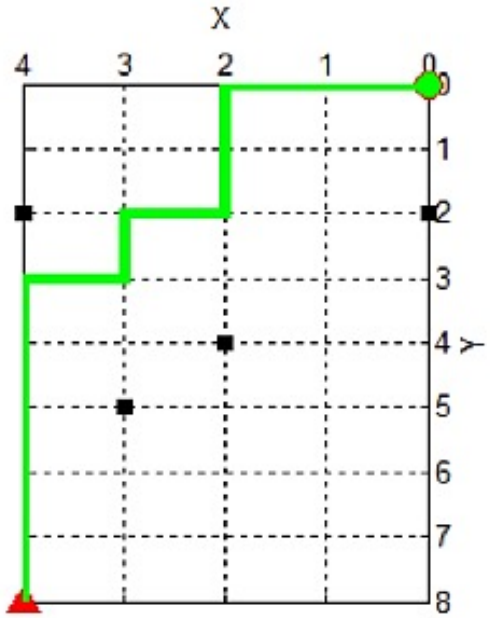
Path Navigation For Indoor Robot With Q-Learning



Path Navigation For Indoor Robot With Q-Learning



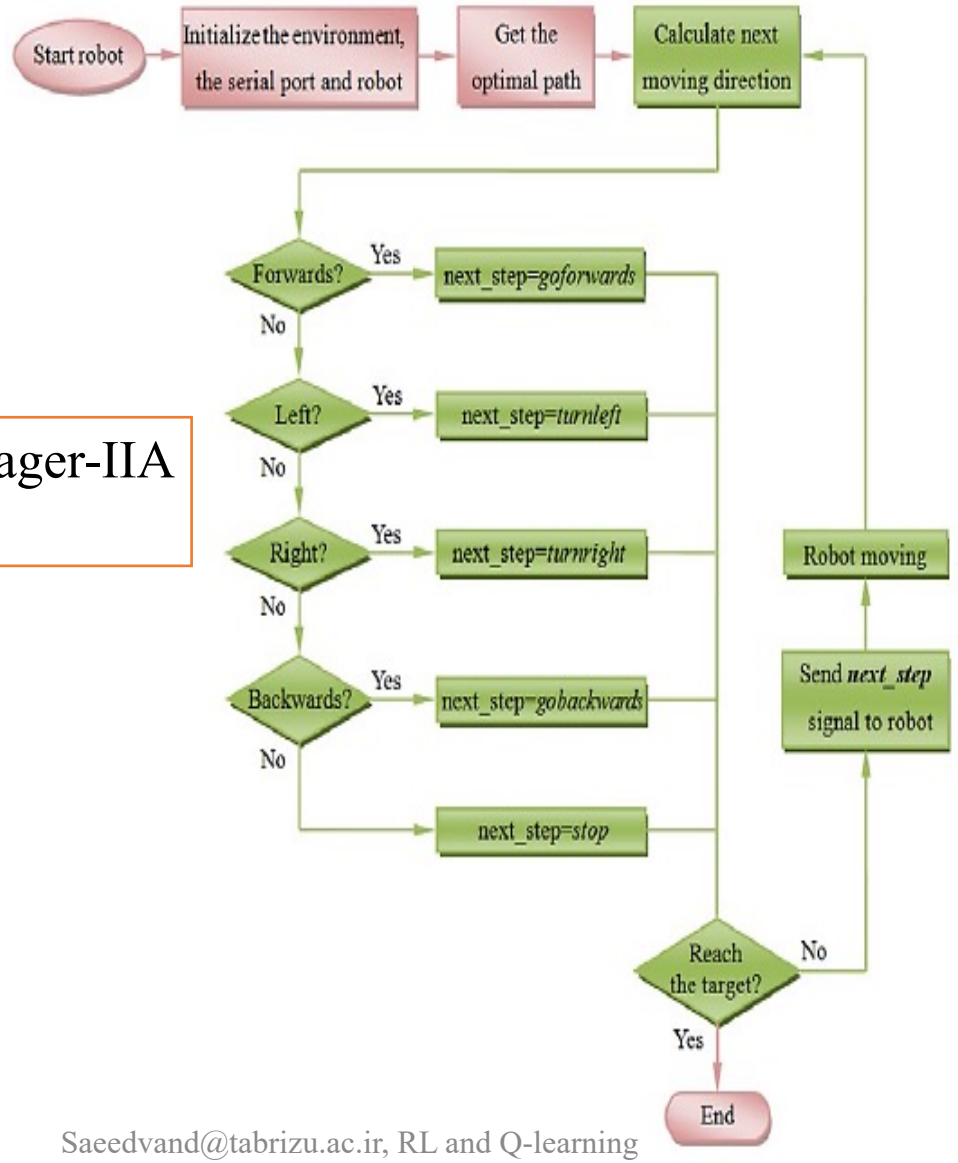
(a) The indoor corridor environment



(b) The grid and topological indoor environment map

Path Navigation For Indoor Robot With Q-Learning

The Workflow of Real Robot Voyager-IIA presented in the paper.

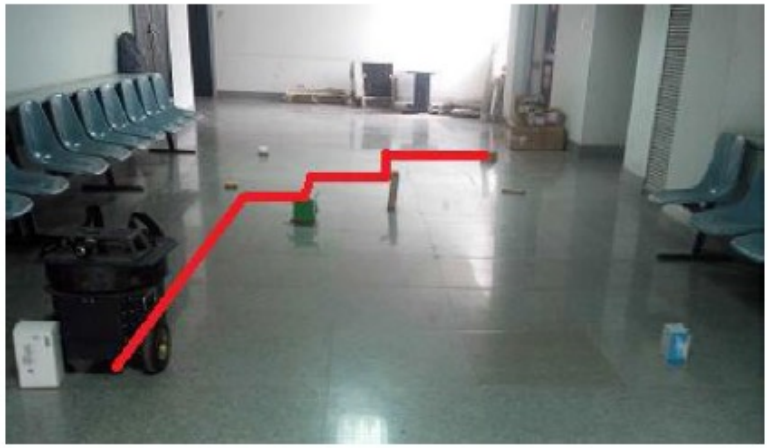


Path Navigation For Indoor Robot With Q-Learning

(a) 15 seconds

(b) 30 seconds

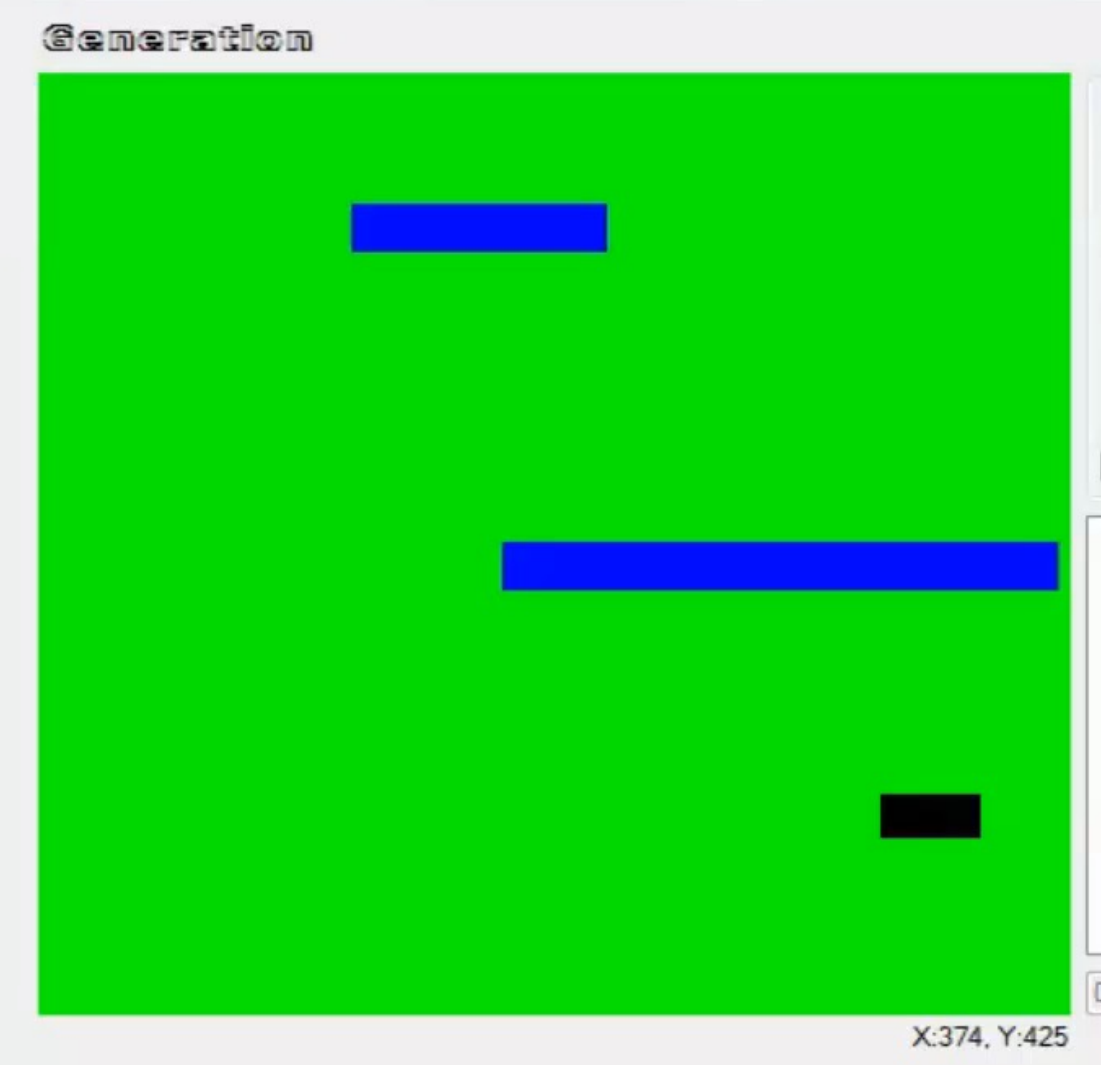
(c) 60 seconds



(d) 90 seconds

Example

- ✓ Multi agent Implementation of Q-learning.



Summery and Conclusion

- **The difference of RL with evolutionary algorithms?**
  - ✓ Environment is dynamic without access to information in advance.

- **Problem with this kind of learning?**
  - ✓ Memory problems if the state-space/environment is large.

- **Solution:**
  - ✓ Using a **Neural Networks** or **Deep learning algorithms** instead of a Q-table for function approximation.