HUMBER

AI for Business Decision Making

AIGC 5503

Case Study: Q-Learning

Case Study: Optimizing the Flows in an E-Commerce Warehouse

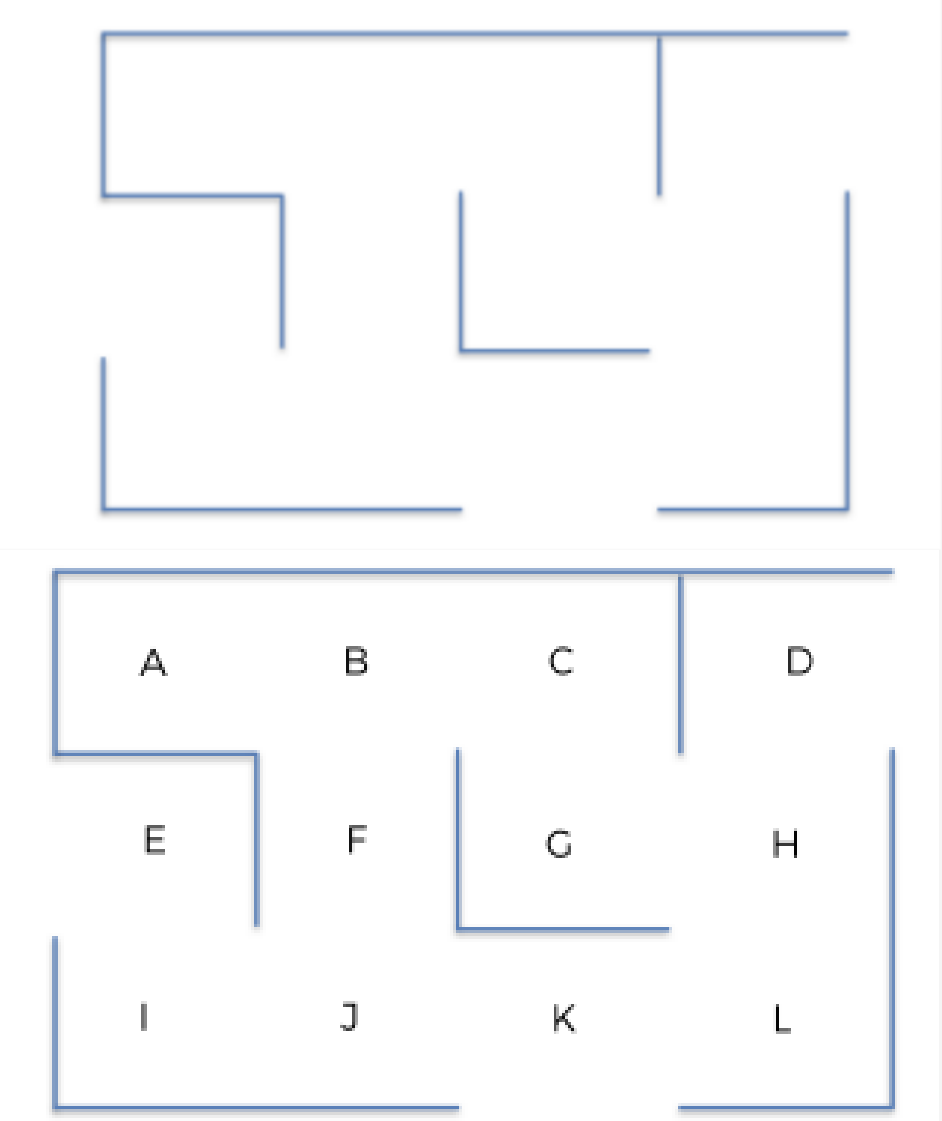
Problem to solve:

Optimize the flows inside an E-Commerce Warehouse

Warehouse Location

The warehouse belongs to an online retail company that sells products to a variety of customers.

Inside this warehouse, the products are stored in 12 different locations, labeled by the letters from A to L as shown.



Plan

1. Defining the Environment

    Defining the states

    Defining the actions

    Defining the rewards

2. AI Solution

    Markov Decision Process

    Temporal Differences

    Q-Learning

3. Implementation in Python

Warehouse Location

As the orders are placed by the customers online, an Autonomous Warehouse Robot is moving around the warehouse to collect the products for future deliveries. Here is what it looks like:

Autonomous Warehouse Robot

Warehouse Location

The 12 locations are all connected to a computer system, which is ranking in real time the priorities of product collection from these 12 locations. For example, at a specific time t, it will return the following ranking:

Priority Rank	Location
1	G
2	K
3	L
4	J
5	A
6	I
7	H
8	C
9	B
10	D
11	F
12	E

1. Environment to define

When building an AI, the first thing we always have to do is to define the environment. And defining an environment always requires all three of the following elements:

- Defining the states
- Defining the actions
- Defining the rewards

Defining the states.

Let's start with the states. The input state is simply the location where our Autonomous Warehouse Robot is at each time t.

However since we will build our AI with mathematical equations, we will encode the locations names (A, B, C,...) into index numbers, with respect to the following mapping:

Location	State
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11

Defining the Actions

The actions are simply the next moves the robot can make to go from one location to the next. So for example, let's say the robot is at location J, the possible actions that the robot can play/move is to go to I, to F, or to K

actions = [0,1,2,3,4,5,6,7,8,9,10,11]

Defining the Rewards

The last thing that we have to do now to build our environment is to define a system of rewards.

More specifically, we have to define a reward function ***r*** that takes as inputs a state ***S*** and an action ***a***, and returns a numerical reward that the AI will get by playing the action ***a*** in the state ***s***:

$$R : (\text{state}, \text{action}) \rightarrow r \in R$$

The whole Q-Learning algorithm

Let's summarize the different steps of the whole Q-Learning process:

Initialization:

For all couples of states s and actions a, the Q-Values are initialized to 0:

$$\forall s \in S, a \in A, Q_0(s,a) = 0$$

We start in the initial state s_0. We play a random possible action and we reach the first state s_1.

Then for each $t \geq 1$, we will repeat for a certain number of times (1000 times in our code) the following:

1. We select a random state s_t from our 12 possible states:

$$s_t = \text{random}(0,1,2,3,4,5,6,7,8,9,10,11)$$

2. We play a random action a_t that can lead to a next possible state, i.e., such that $R(s_t, a_t) > 0$:

$$a_t = \text{random}(0,1,2,3,4,5,6,7,8,9,10,11) \text{ s.t. } R(s_t, a_t) > 0$$

3. We reach the next state s_{t+1} and we get the reward $R(s_t, a_t)$

4. We compute the Temporal Difference $TD_t(s_t, a_t)$:

$$TD_t(s_t, a_t) = R(s_t, a_t) + \gamma \max_a(Q(s_{t+1}, a)) - Q(s_t, a_t)$$

5. We update the Q-value by applying the Bellman equation:

$$Q_t(s_t, a_t) = Q_{t-1}(s_t, a_t) + \alpha TD_t(s_t, a_t)$$