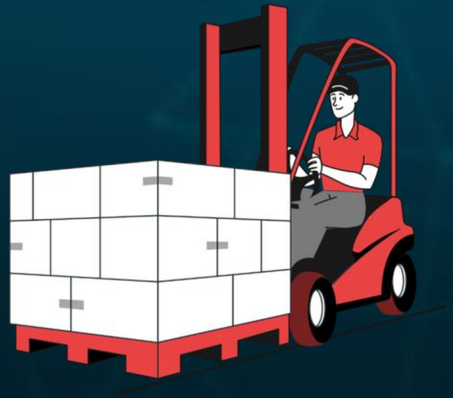
Case of Study – Optimizing the Flows in an E-Commerce Warehouse

Luan Santos
July, 2022

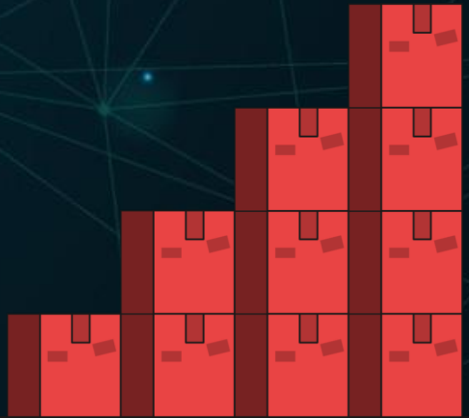


Outline

Executive Summary

This case study aims to illustrate the creation of an algorithm, using Q-Learning, capable of informing the most efficient route between two points, so that the greatest time savings in terms of displacement can be achieved, something with a lot of focus on the logistics sector.

First, the formula behind the process of creating the AI used and its various variables was demonstrated. So, based on the contents of the **Artificial Intelligence for Business** course, a function was created capable of generating the best route between two points. However, as the function had a limitation with regard to the number of points to be used to trace the route, a new function was built on top of the first one, which allows the process to be interactive, without the need to change the code. with the specific number of points you want to analyze.



Introduction

Background and Context

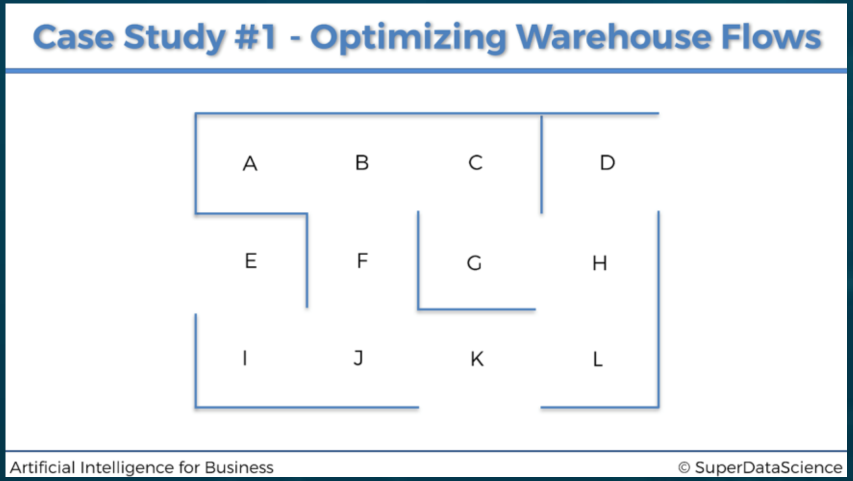
- One of the classic business challenges that can be found refers to the issue of flow logistics. A company with good flow optimizes its processes and avoids traffic and unnecessary spending of time, which is reflected in monetary savings.

Problem at Hand

- It was given the task of optimizing the movement flow in a technology company's warehouse, as a way of optimizing time.

Solution

- As a way of solving the question that was imposed on us, an algorithm will be created capable of generating the best route to be followed (in a predetermined environment) between two points, including intermediate points, as a way of optimizing the time spent in displacement. For this, reinforcement learning through Q-Learning will be used, based on the Bellman equation.


Case Study #1 - Optimizing Warehouse Flows
A
B
C
D
E
F
G
H
I
J
K
L
Artificial Intelligence for Business
© SuperDataScience

Section 1
Methodology

Methodology

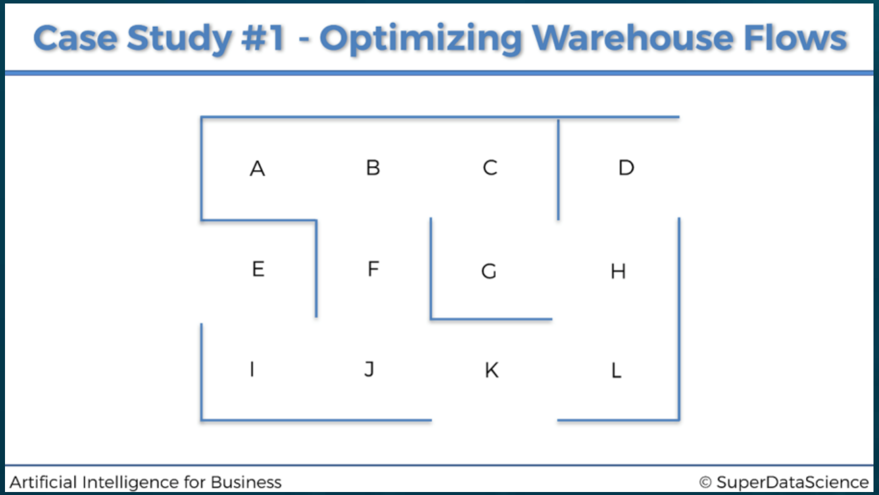
Summary

- 1° - Set the Environment
- 2° - Build the solution in artificial intelligence with Q-Learning
- 3° - Model into production

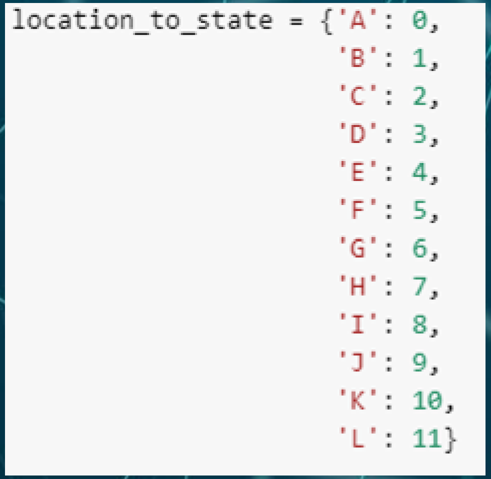
Defining The Environment

In order to define the environment, it is necessary to consider all the possible movements that can be performed within it (in this case, the warehouse).

As can be seen in the figure on the side, there are a total of 12 spaces that can be considered (A to L) .


Case Study #1 - Optimizing Warehouse Flows
A
B
C
D
E
F
G
H
I
J
K
L
Artificial Intelligence for Business
© SuperDataScience

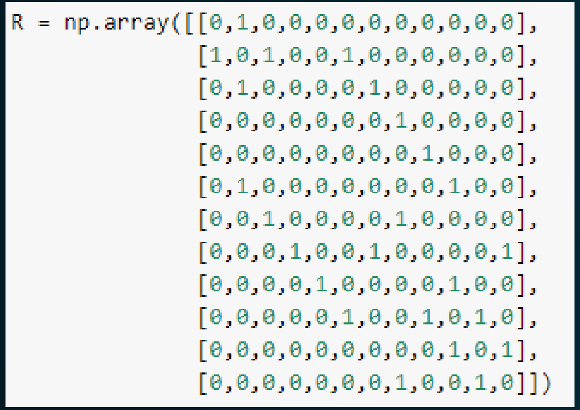
Defining The Environment - Defining the states



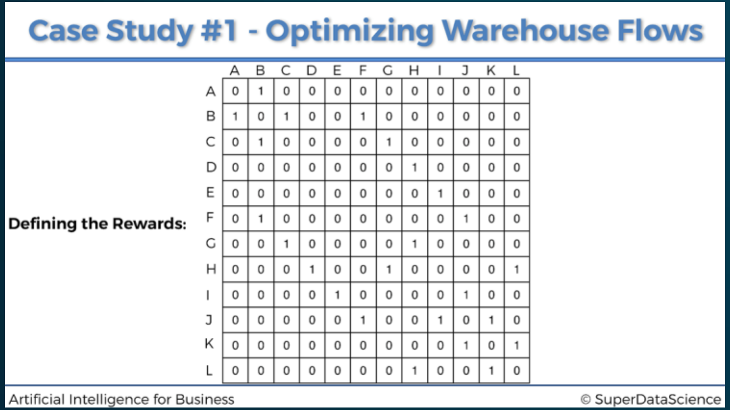
```
location_to_state = {'A': 0,
                     'B': 1,
                     'C': 2,
                     'D': 3,
                     'E': 4,
                     'F': 5,
                     'G': 6,
                     'H': 7,
                     'I': 8,
                     'J': 9,
                     'K': 10,
                     'L': 11}
```

Thus, each space will be numbered in a dictionary so that they can be worked on in the construction of the AI.

Defining The Environment - Defining the rewards



```
R = np.array([[0,1,0,0,0,0,0,0,0,0,0,0],
              [1,0,1,0,0,1,0,0,0,0,0,0],
              [0,1,0,0,0,0,1,0,0,0,0,0],
              [0,0,0,0,0,0,0,1,0,0,0,0],
              [0,0,0,0,0,0,0,0,1,0,0,0],
              [0,1,0,0,0,0,0,0,0,1,0,0],
              [0,0,1,0,0,0,0,1,0,0,0,0],
              [0,0,0,1,0,0,1,0,0,0,0,1],
              [0,0,0,0,1,0,0,0,0,1,0,0],
              [0,0,0,0,0,1,0,0,1,0,1,0],
              [0,0,0,0,0,0,0,0,0,1,0,1],
              [0,0,0,0,0,0,0,1,0,0,1,0]])
```



Case Study #1 - Optimizing Warehouse Flows

Defining the Rewards:

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	0	0	0	0	0	0	0	0	0	0
B	1	0	1	0	0	1	0	0	0	0	0	0
C	0	1	0	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	0	0	1	0	0	0	0
E	0	0	0	0	0	0	0	0	1	0	0	0
F	0	1	0	0	0	0	0	0	0	1	0	0
G	0	0	1	0	0	0	0	1	0	0	0	0
H	0	0	0	1	0	0	1	0	0	0	0	1
I	0	0	0	0	1	0	0	0	0	1	0	0
J	0	0	0	0	0	1	0	0	1	0	1	0
K	0	0	0	0	0	0	0	0	0	1	0	1
L	0	0	0	0	0	0	0	1	0	0	1	0

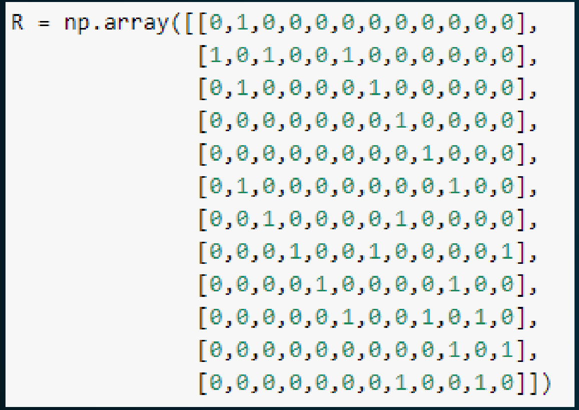
Artificial Intelligence for Business © SuperDataScience

Another important point to be considered for the application of the Bellman equation is what refers to the rewards.

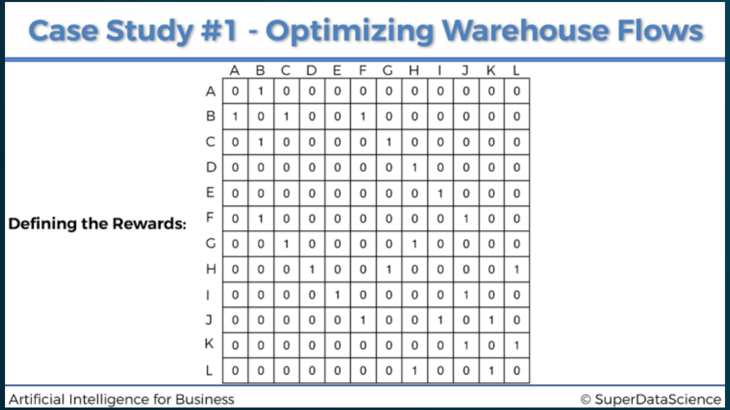
As to go from one point to another it is necessary to follow a sequence of spaces to be covered, a reward matrix was created in which a positive score was given to possible movements to be performed (example: going directly from point F to point B or J) and zero points for impossible moves (example: going directly from point F to point K).

NOTE: in cases where there is more than one path to go from one point to another, the algorithm randomly traces one of the paths that can be traversed. In order for one path to be preferred over another, it is necessary to give greater value in the reward matrix to the desired path. For example, to go from J to G, you can go by F or K; if there is a preference to pass through F, it is necessary to give greater reward to the element of the matrix that corresponds to this transition (in this case, the element of row 10 and column 6).

Defining The Environment - Defining the rewards



```
R = np.array([[0,1,0,0,0,0,0,0,0,0,0,0],
              [1,0,1,0,0,1,0,0,0,0,0,0],
              [0,1,0,0,0,0,1,0,0,0,0,0],
              [0,0,0,0,0,0,0,1,0,0,0,0],
              [0,0,0,0,0,0,0,0,1,0,0,0],
              [0,1,0,0,0,0,0,0,0,1,0,0],
              [0,0,1,0,0,0,0,1,0,0,0,0],
              [0,0,0,1,0,0,1,0,0,0,0,1],
              [0,0,0,0,1,0,0,0,0,1,0,0],
              [0,0,0,0,0,1,0,0,1,0,1,0],
              [0,0,0,0,0,0,0,0,0,1,0,1],
              [0,0,0,0,0,0,0,1,0,0,1,0]])
```



Case Study #1 - Optimizing Warehouse Flows

Defining the Rewards:

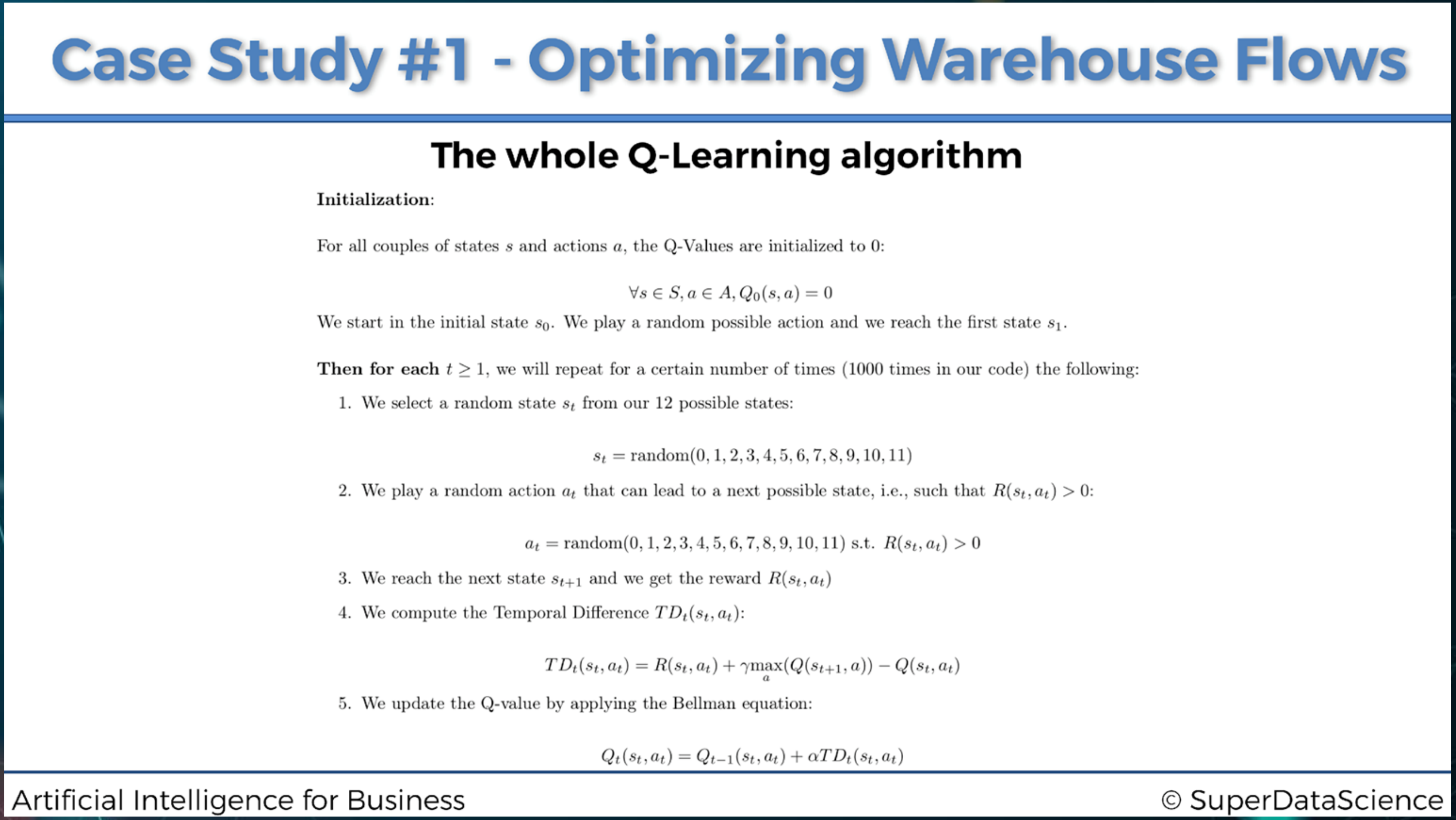
	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	0	0	0	0	0	0	0	0	0	0
B	1	0	1	0	0	1	0	0	0	0	0	0
C	0	1	0	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	0	0	1	0	0	0	0
E	0	0	0	0	0	0	0	0	1	0	0	0
F	0	1	0	0	0	0	0	0	0	1	0	0
G	0	0	1	0	0	0	0	1	0	0	0	0
H	0	0	0	1	0	0	1	0	0	0	0	1
I	0	0	0	0	1	0	0	0	0	1	0	0
J	0	0	0	0	0	1	0	0	1	0	1	0
K	0	0	0	0	0	0	0	0	0	1	0	1
L	0	0	0	0	0	0	0	1	0	0	1	0

Artificial Intelligence for Business © SuperDataScience

NOTE: in cases where there is more than one path to go from one point to another, the algorithm randomly traces one of the paths that can be traversed. In order for one path to be preferred over another, it is necessary to give greater value in the reward matrix to the desired path. For example, to go from J to G, you can go by F or K; if there is a preference to pass through F, it is necessary to give greater reward to the element of the matrix that corresponds to this transition (in this case, the element of row 10 and column 6).

Building The Ai Solution With Q-learning

The construction of the solution in Q-Learning is based on the Bellman equation, which can be seen in the figure to the side.



Case Study #1 - Optimizing Warehouse Flows

The whole Q-Learning algorithm

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 $TD_t(s_t,a_t)$:

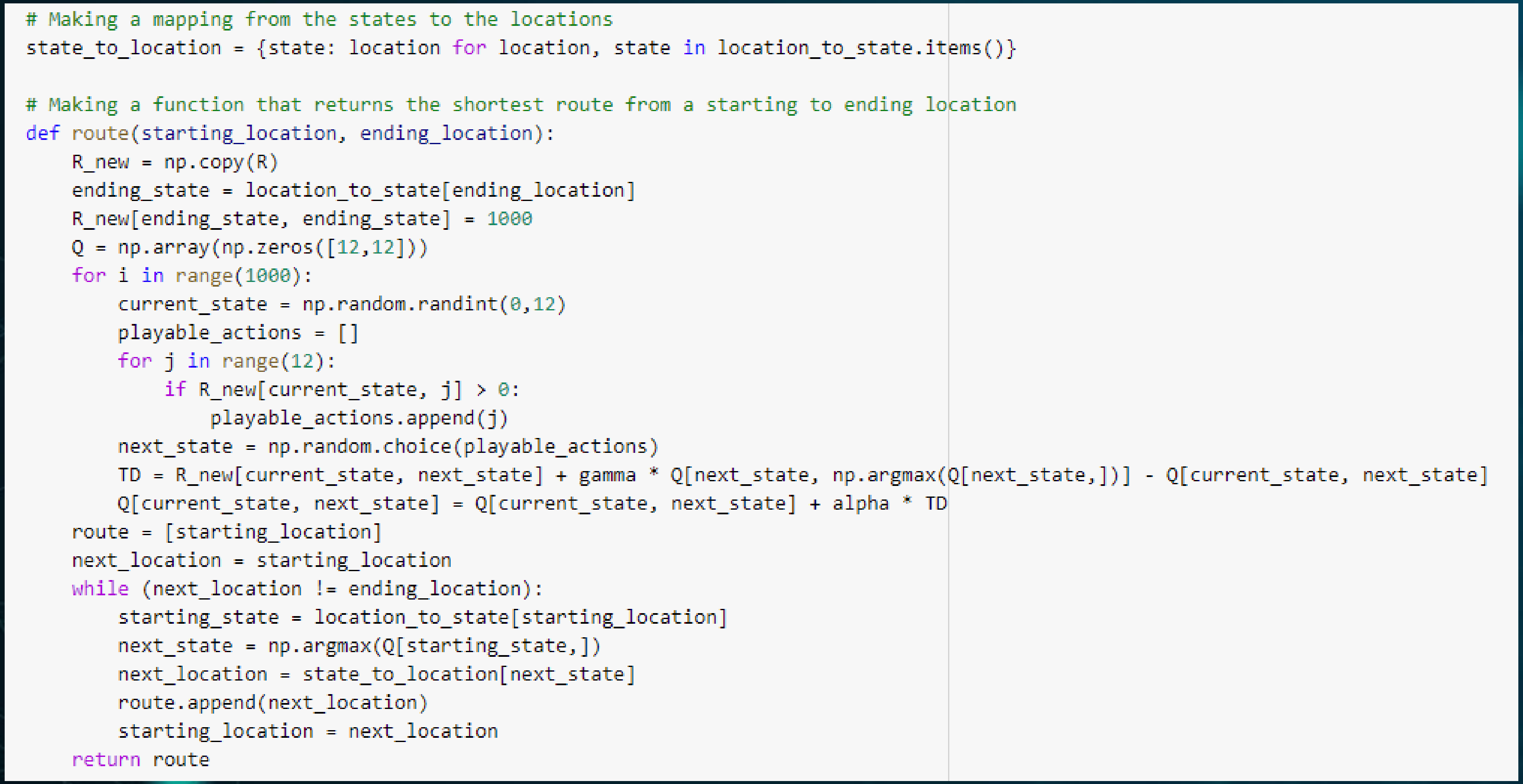
$$TD_t(s_t,a_t) = R(s_t,a_t) + \gamma \max_a(Q(s_{t+1},a)) - Q(s_t,a_t)$$

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

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

Artificial Intelligence for Business © SuperDataScience

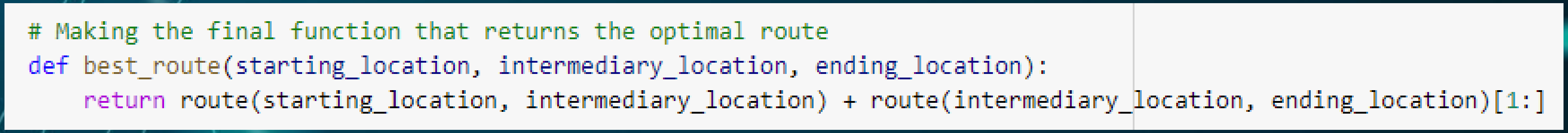
Building The Ai Solution With Q-learning and Going Into Production



```python
# Making a mapping from the states to the locations
state_to_location = {state: location for location, state in location_to_state.items()}

# Making a function that returns the shortest route from a starting to ending location
def route(starting_location, ending_location):
    R_new = np.copy(R)
    ending_state = location_to_state[ending_location]
    R_new[ending_state, ending_state] = 1000
    Q = np.array(np.zeros([12,12]))
    for i in range(1000):
        current_state = np.random.randint(0,12)
        playable_actions = []
        for j in range(12):
            if R_new[current_state, j] > 0:
                playable_actions.append(j)
        next_state = np.random.choice(playable_actions)
        TD = R_new[current_state, next_state] + gamma * Q[next_state, np.argmax(Q[next_state,])] - Q[current_state, next_state]
        Q[current_state, next_state] = Q[current_state, next_state] + alpha * TD
    route = [starting_location]
    next_location = starting_location
    while (next_location != ending_location):
        starting_state = location_to_state[starting_location]
        next_state = np.argmax(Q[starting_state,])
        next_location = state_to_location[next_state]
        route.append(next_location)
        starting_location = next_location
    return route
```

Going Into Production - Intermediary Point



```
# Making the final function that returns the optimal route
def best_route(starting_location, intermediary_location, ending_location):
    return route(starting_location, intermediary_location) + route(intermediary_location, ending_location)[1:]
```

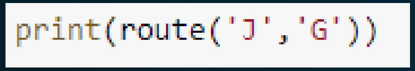
Finally, to make it possible to trace routes, considering an intermediate point, the ***best_route*** function was created, which uses the route function as a base. However, it has a certain limitation, because in order to add new intermediate points, it is necessary to change the line of code.

Section 2
Results

Results – Route between 2 points

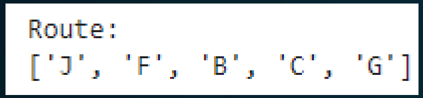
J to G

Input:



```
print(route('J','G'))
```

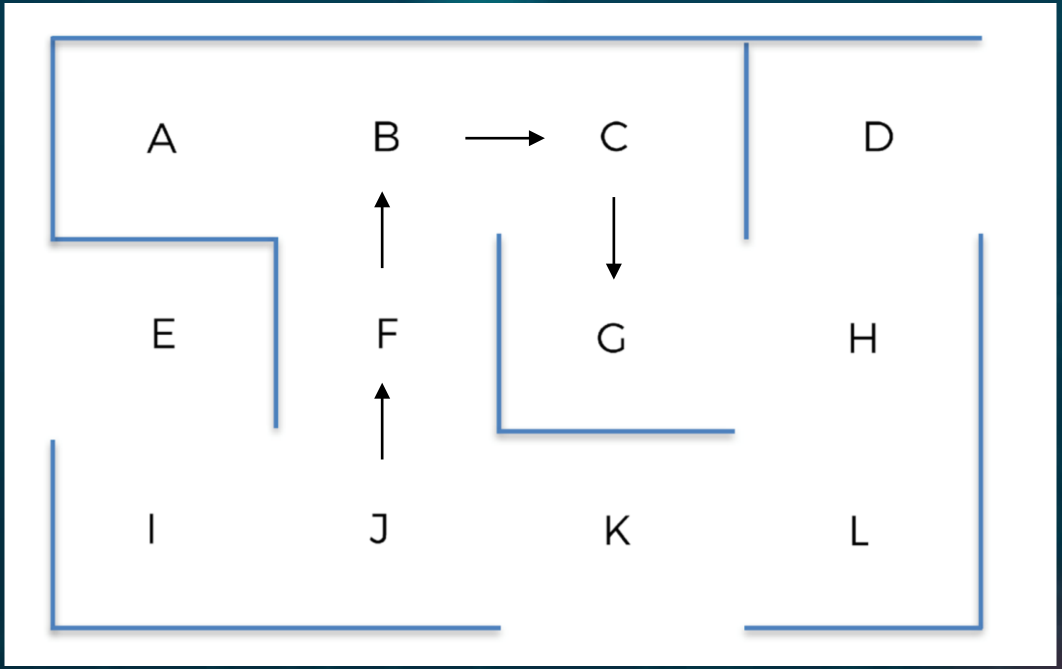
Output:



```
Route:
['J', 'F', 'B', 'C', 'G']
```

In this case, to go from J to G, the algorithm chose to follow the path through which point F is. As the value of the reward in going through F or K is the same, occasionally the path through K would be chosen when rotating the line code more often.

For the algorithm to prefer the path through K, it would be necessary for the transition from J to K to have a greater reward than the transition from J to F.



Results – Route between 3 or more points

D to F to G

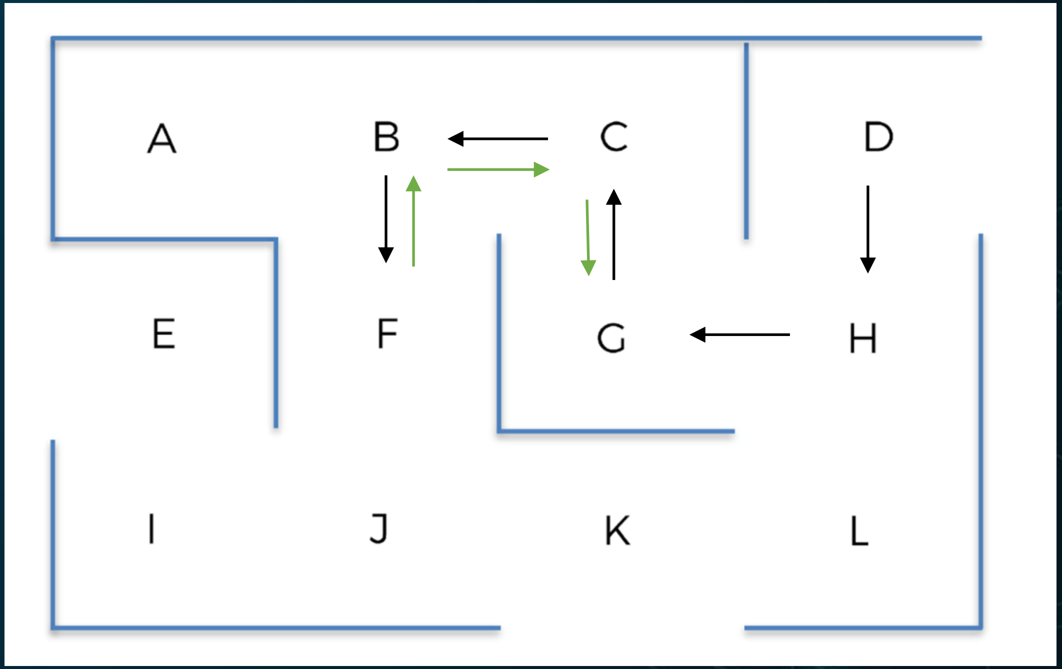
Input:

```
print(best_route('D', 'F', 'G'))
```

Output:

```
Route:
['D', 'H', 'G', 'C', 'B', 'F', 'B', 'C', 'G']
```

As in the previous case, the algorithm chose to go through G for the route between D and F. As the distance to pass through L is the same, when running the line of code more times, this path appears as an option. In order for there to be a preference for paths, it is necessary to increase the reward for the desired path.

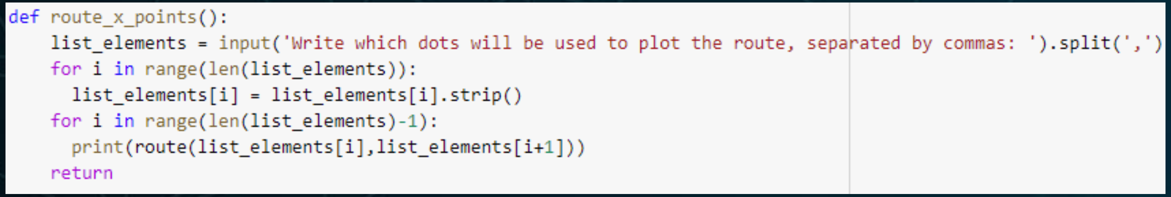


Section 3
Other Scenarios

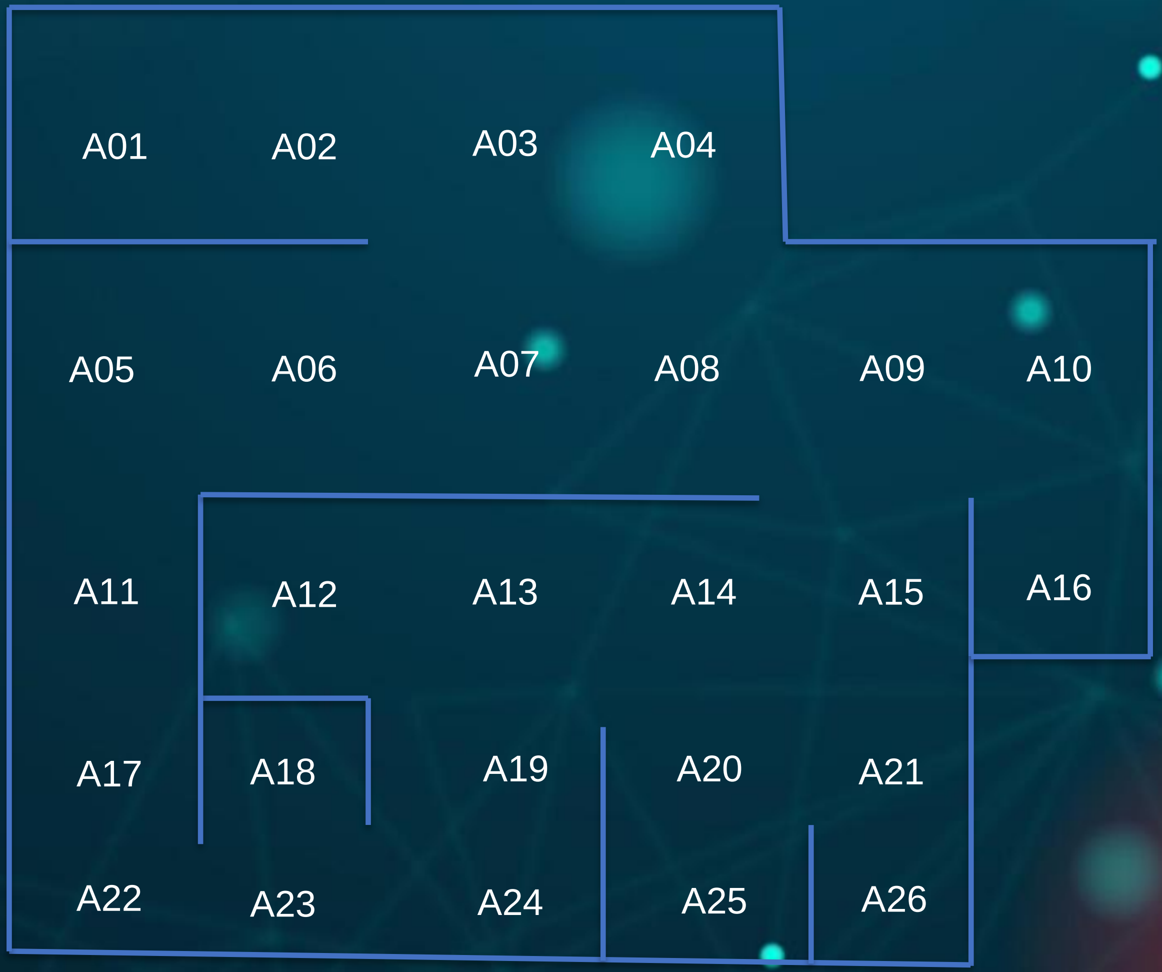
Other Scenarios - New Warehouse

As a way of exploring this technique, it will be adopted in other scenarios, such as the warehouse on the right, which represents a more complex environment than the one shown above.

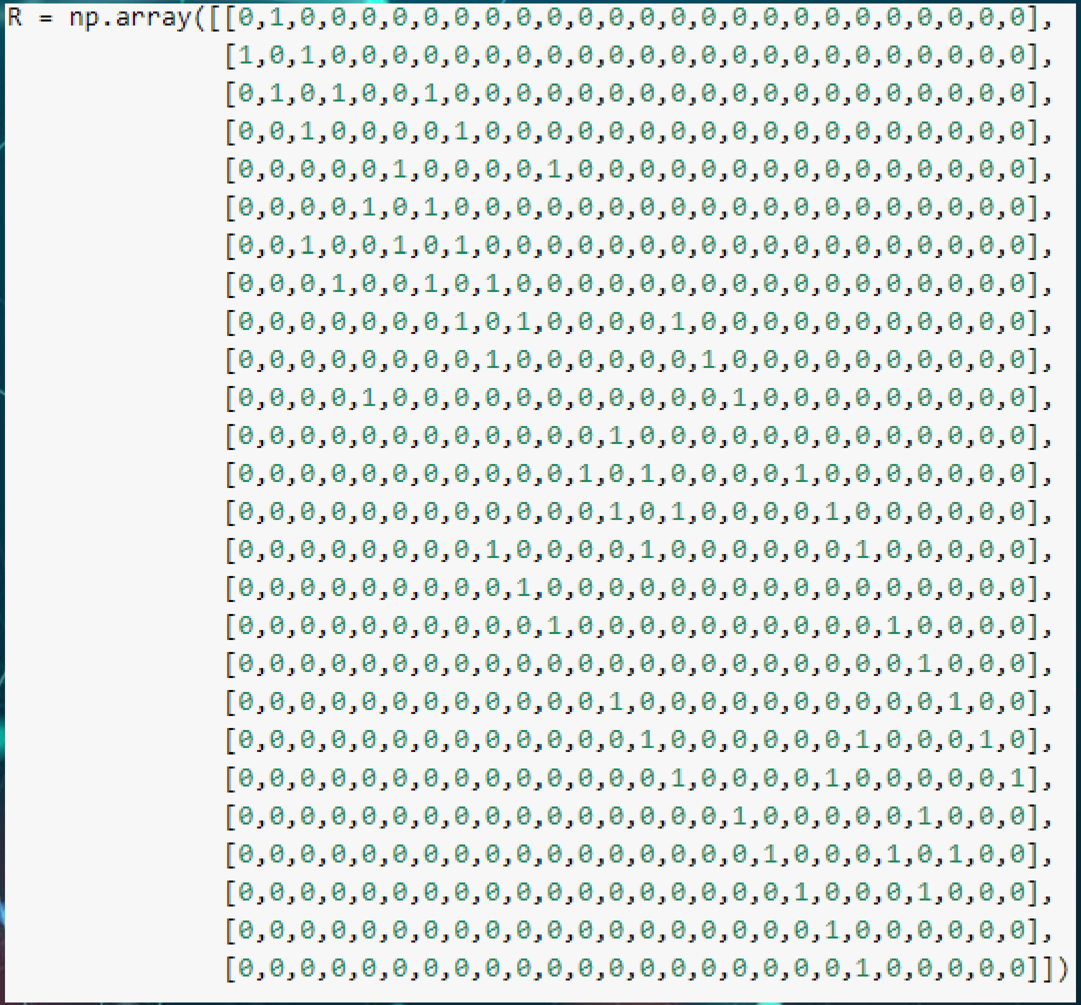
In addition, a function was created (***route_x_points***) that allows the process of tracing the route between points to be done in an automated way, without the need to be restricted to a specific number of points, which represents an update of the ***route*** function previous.



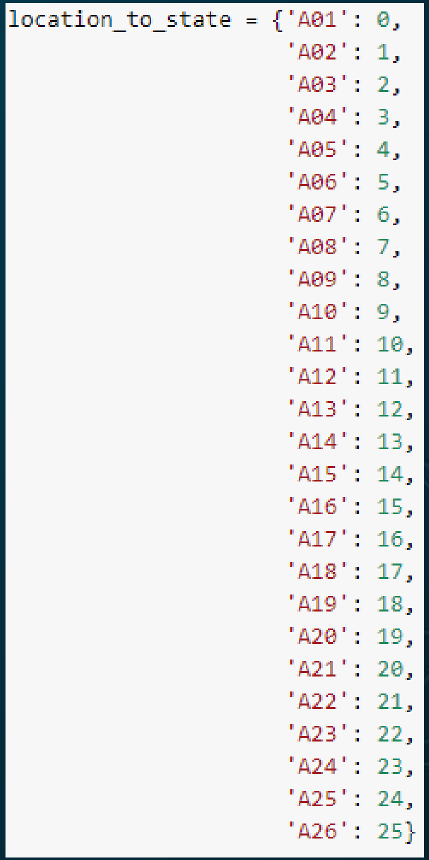
```
def route_x_points():
    list_elements = input('Write which dots will be used to plot the route, separated by commas: ').split(',')
    for i in range(len(list_elements)):
      list_elements[i] = list_elements[i].strip()
    for i in range(len(list_elements)-1):
      print(route(list_elements[i],list_elements[i+1]))
    return
```



Other Scenarios - New Reward Matrix and States



```
R = np.array([[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]])
```



```
location_to_state = {'A01': 0,
                     'A02': 1,
                     'A03': 2,
                     'A04': 3,
                     'A05': 4,
                     'A06': 5,
                     'A07': 6,
                     'A08': 7,
                     'A09': 8,
                     'A10': 9,
                     'A11': 10,
                     'A12': 11,
                     'A13': 12,
                     'A14': 13,
                     'A15': 14,
                     'A16': 15,
                     'A17': 16,
                     'A18': 17,
                     'A19': 18,
                     'A20': 19,
                     'A21': 20,
                     'A22': 21,
                     'A23': 22,
                     'A24': 23,
                     'A25': 24,
                     'A26': 25}
```

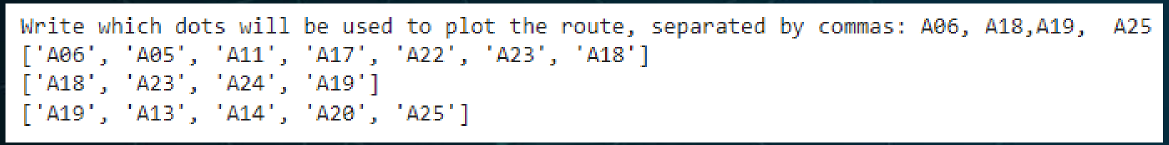
Naturally, it was necessary to change the rewards matrix in order to indicate all the movements that are possible to be performed, in addition to updating the states with the existing total points.

Other Scenarios - Result

A06 to A18 to A19 to A25

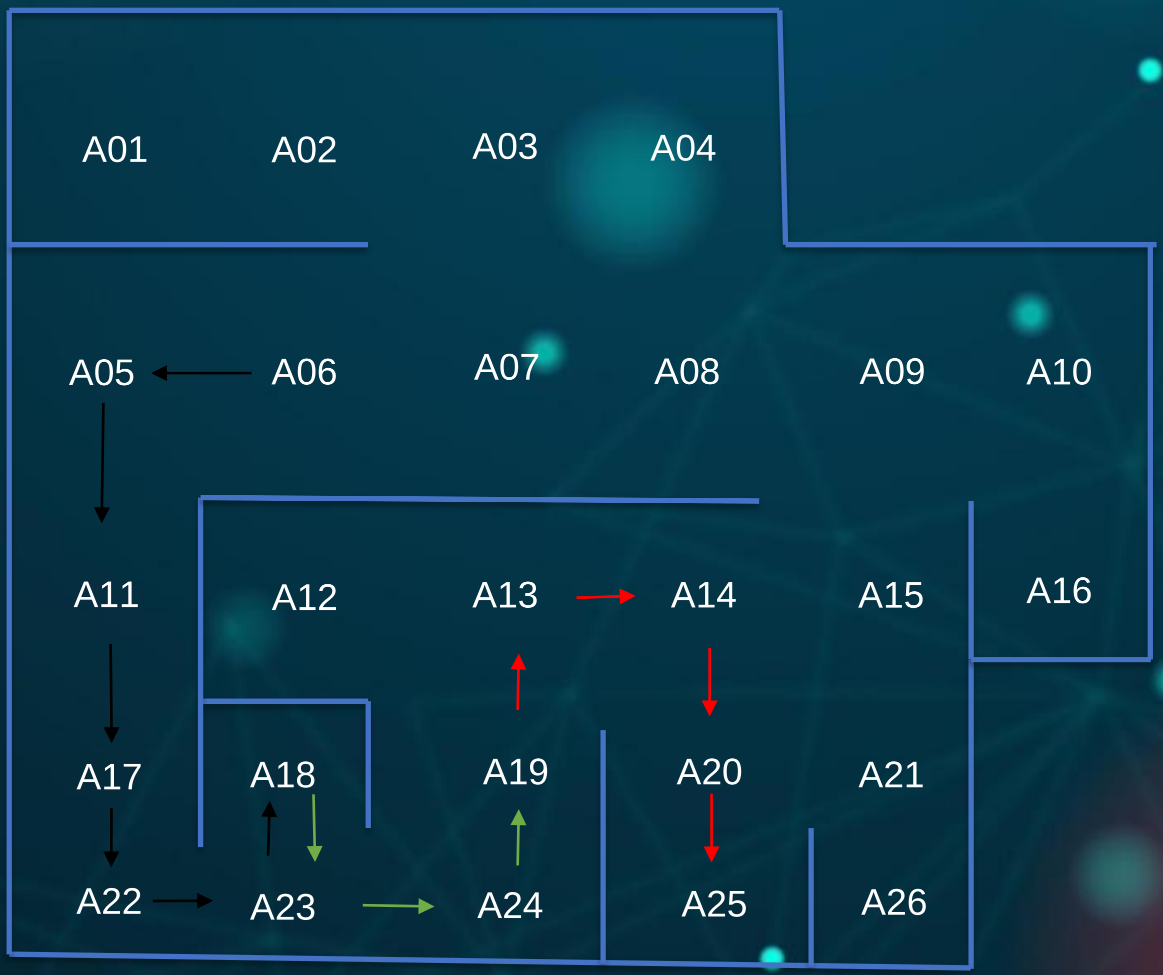
Input: `route_x_points()`

Output:



```
Write which dots will be used to plot the route, separated by commas: A06, A18,A19,  A25
['A06', 'A05', 'A11', 'A17', 'A22', 'A23', 'A18']
['A18', 'A23', 'A24', 'A19']
['A19', 'A13', 'A14', 'A20', 'A25']
```

With the new function, you can insert as many intermediate points as you want, without having to change the line of code, which provides great freedom.



Section 4
Conclusion

Conclusion

As it was possible to verify, the process of generating routes as a way of optimizing business and logistics processes can be applied in the most diverse scenarios, requiring only the states, reward matrix and a few values to be changed in the creation of the AI.

Appendix

This case study could not have existed without the **Artificial Intelligence for Business** course. Therefore, a special thanks to professors Hadelin de Ponteves and Kirill Eremenko.

https://www.udemy.com/course/ai-for-business/