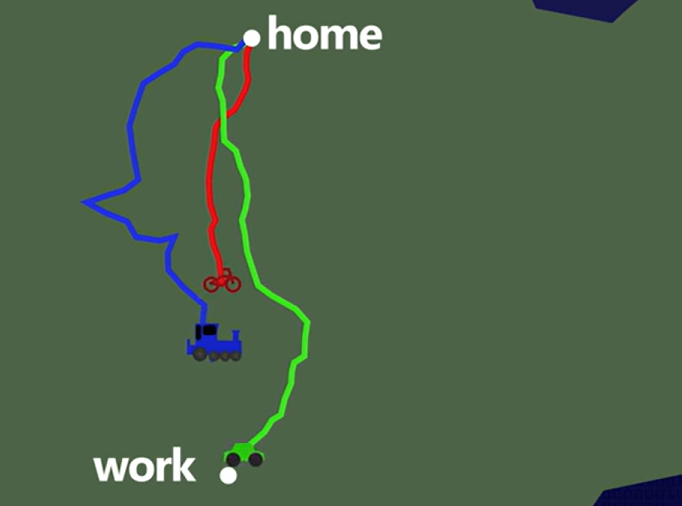
Markov Decision Process

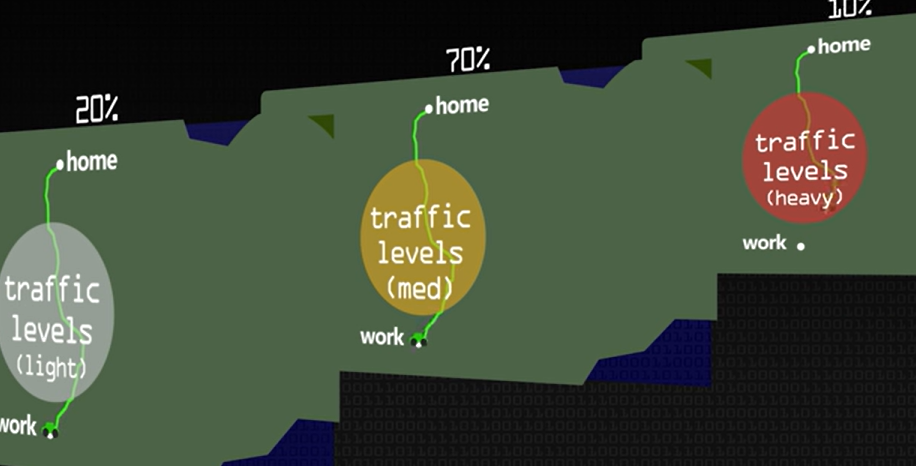


Home to Work

We can go by car,bike,train

Home,work=>states

Car,bike,train=>action



Markov is probabilitic assumption it is just a state transition probability of outcome depend on current state does not depend on history …..

The only thing that determines the probability is action

In markov decision process we cant take a path ….When we take a step the world changes probabilitically stochastically

So instead of path have to use policy .Policy is a lookup table what is the optimal action ( best action ) to take to achieve particular specification  
  
  
Transition probabilities:

State 0 State 0 State 1 State 2  
  
Action 1  
  
Action 2  
  
State 1 State 0 State 1 State 2  
  
Action 1  
  
Action 2  
  
State 2 State 0 State 1 State 2  
  
Action 1  
  
Action 2  
  
[

[

[0.26446561 0.27811243 0.45742195]

[0.36116871 0.3535665 0.28526479]

]

[

[0.26532922 0.51717793 0.21749285]

[0.28261936 0.36520618 0.35217446]

]

[

[0.28978286 0.40501022 0.30520692]

[0.44302559 0.30947724 0.24749717]

]

]

Rewards:

* In state 0 (e.g., being in the middle lane), taking action 1 (e.g., changing lanes to the left) might incur a negative reward if it leads to a collision or traffic jam.
* Conversely, taking action 2 (e.g., changing lanes to the right) might result in a positive reward if it leads to a faster route or smoother traffic flow.

Discount factor:

* **discount\_factor**: A discount factor (usually denoted as gamma) used to discount future rewards. It's a value between 0 and 1 (default is 0.9).
* **values**: An array storing the estimated values (expected cumulative rewards) for each state. It's initialized as an array of zeros with length **num\_states**.
* **num\_states = 3**
* **num\_actions = 2**
* **transition\_probabilities**:

[[[0.8, 0.2], [0.3, 0.7], [0.1, 0.9]],

[[0.6, 0.4], [0.9, 0.1], [0.4, 0.6]],

[[0.2, 0.8], [0.5, 0.5], [0.7, 0.3]]]

* **rewards:**

[[[1, 2], [0, -1], [2, 1]],

[[-1, 1], [3, 0], [1, -2]],

[[2, -1], [0, 1], [-1, 2]]]

* **discount\_factor = 0.9**

Let's focus on updating the Q-values for state 0.

1. **Initialize Q-values for Actions**:
   * For state 0, initialize an empty list **q\_values** to store Q-values for each action.
   * Iterate over actions (0 and 1).
2. **Calculate Q-values for Each Action**:
   * For action 0:
     + Iterate over next states (0, 1, 2).
     + Calculate the Q-value using the formula:

Q[state, action] = transition\_probabilities[state, action, next\_state] \* (rewards[state, action, next\_state] + discount\_factor \* values[next\_state])

Sum the Q-values obtained for each next state.

* + For action 1, repeat the same process.

**3.Select Maximum Q-value**:

* + Choose the action associated with the maximum Q-value.

**4.Update State Value**:

* + Set the value of state 0 to the maximum Q-value obtained.

**5.Repeat for Other States**:

* + Repeat the above steps for states 1 and 2.

Let's demonstrate this process for state 0:

1. **For State 0**:
   * For action 0:
     + Calculate Q-values for transitioning to states 0, 1, and 2.
       - Q-value for transitioning to state 0: **0.8 \* (1 + 0.9 \* 0) = 0.8**
       - Q-value for transitioning to state 1: **0.3 \* (0 - 1 \* 0) = -0.3**
       - Q-value for transitioning to state 2: **0.1 \* (2 + 0.9 \* 0) = 0.2**
     + Sum of Q-values: **0.8 + (-0.3) + 0.2 = 0.7**
   * For action 1:
     + Calculate Q-values for transitioning to states 0, 1, and 2.
       - Q-value for transitioning to state 0: **0.2 \* (2 + 0.9 \* 0) = 0.4**
       - Q-value for transitioning to state 1: **0.7 \* (-1 + 0.9 \* 0) = -0.7**
       - Q-value for transitioning to state 2: **0.9 \* (1 + 0.9 \* 0) = 0.9**
     + Sum of Q-values: **0.4 + (-0.7) + 0.9 = 0.6**
   * Maximum Q-value: **0.7**
   * Update state value: **values[state] = 0.7**

Values=[ state 0 q-value , state 1 q-value , state 2 q-value ]

* Suppose **car\_B.distance** is 10 units.
* If **agent.num\_states** is 3, the maximum allowed state index would be 22 (since the indices start from 0).
* The line would then evaluate as **state = min(max(0, 10), 2)**, which becomes **state = min(10, 2)**, resulting in **state = 2**.
* If **car\_B.distance** is -5 units, the line would evaluate as **state = min(max(0, -5), 2)**, which becomes **state = min(0, 2)**, resulting in **state = 0**.

**self.values** is **[0.7, 0.8, 0.9]**, and calling **np.argmax(self.values[0])** will return the index of the maximum value in **[0.7, 0.8, 0.9]**, which is 2