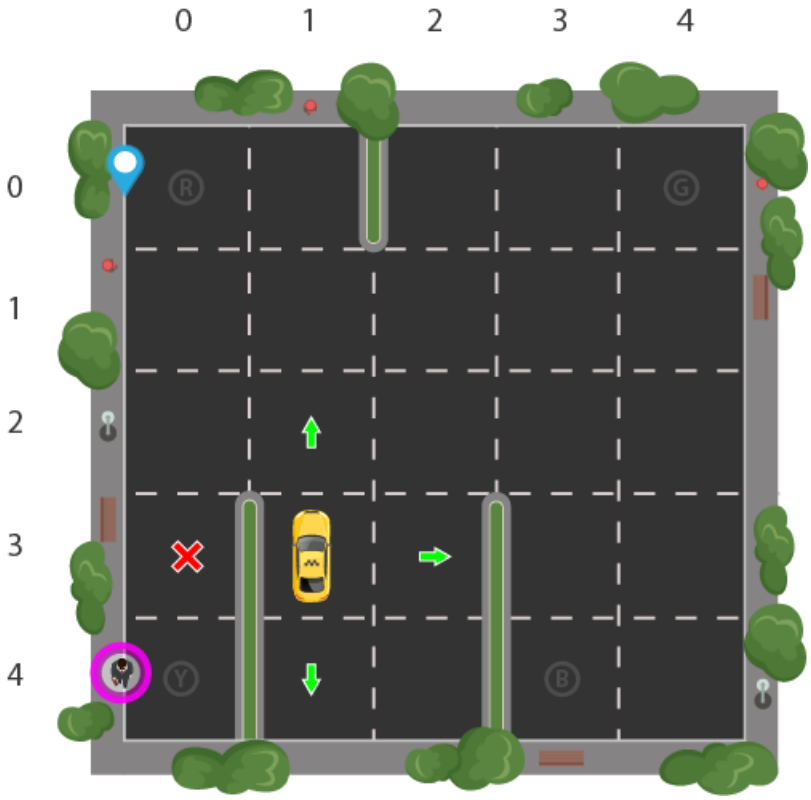
Aprendizagem Reforçada

Taxi Auto-Dirigível

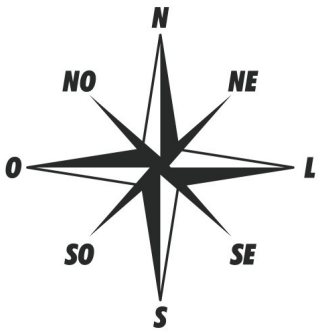
O agente de Aprendizagem Reforçada *(Reinforcement Learning)* encontra um estado/cenário, e então toma uma ação de acordo com o estado/cenário atual. O objetivo é que o agente aprenda a pegar o passageiro numa posição e deixá-lo no destino.



- 5 x 5 = 25 possíveis posições
- Posição atual do táxi (3,1) - Linha 3 e Coluna 1
- 4 locais para pegar (pick up) e deixar (drop off) passageiros: R, G, Y B
- Locais
  - R (0,0)
  - G (0,4)
  - Y (4,0)
  - B (4,3)
- Posição do Passageiro sempre estará em AZUL
- O destino do passageiro estará sempre em ROSA
- Logo, no cenário atual o passageiro está no Y e deseja chegar no R
- As possíveis posições do passageiro são os 4 locais, mais 1 da posição de dentro do táxi
- Se a gente contabilizar todas as possíveis posições, teremos:
  - Posições do Táxi (5x5) | 5 Posições do Passageiro | 4 destinos
  - 5 x 5 x 5 x 4 = 500 estados/cenários possíveis

6 Possíveis Ações

1. South (Sul)
2. North (Norte)
3. East (Leste)
4. West (Oeste)
5. Pickup (Pegar)
6. Dropoff (Deixar)



In [1]:

```
# pip install gym
import gym
```

In [2]:

```
env = gym.make('Taxi-v3').env
env.render()
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

O objetivo é pegar um passageiro em um local e deixá-lo no destino requerido.

- **Recebe +20 pontos** quando deixar um passageiro no destino correto
- **Perder -1 ponto** a cada movimento que ele dá
- **Perde -10 pontos** para tentativa de pegar ou deixar o passageiro numa posição ilegal

O agente aprende a fazer 6 ações de 0-5

- 0 = south
- 1 = north
- 2 = east
- 3 = west
- 4 = pickup
- 5 = dropoff

PS1: Quando o táxi está com um passageiro dentro, sua cor muda para **verde**.
PS2: Os 500 estados possíveis, numerados de 0-499 são uma codificação da posição do táxi, passageiro, e destino.
PS3: O táxi não consegue fazer nenhum movimento em direção a parede. Quando o faz, **perde -1** e não sai do lugar.

In [3]:

```
env.reset()
env.render()

print('Ações possíveis', env.action_space)
print('Posições possíveis', env.observation_space)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

Ações possíveis Discrete(6)
Posições possíveis Discrete(500)
```

Index	Letra
0	R
1	G
2	Y
3	B

Renderizar o cenário da ilustração

In [5]:

```
estado = env.encode(3,1,0,2) # taxi linha, taxi coluna, idx destino, idx passageiro
print('Estado', estado)
env.s = estado
env.render()
```

```
Estado 322
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Tabela da Recompensa

{ação: [(probabilidade, próximo_estado, recompensa, terminado)]
ação: [(probabilidade, próximo_estado, recompensa, terminado)]
ação: [(probabilidade, próximo_estado, recompensa, terminado)]
ação: [(probabilidade, próximo_estado, recompensa, terminado)]
ação: [(probabilidade, próximo_estado, recompensa, terminado)]
ação: [(probabilidade, próximo_estado, recompensa, terminado)]}

In [6]:

```
env.P[322]
```

Out[6]:

```
{0: [(1.0, 422, -1, False)],
 1: [(1.0, 222, -1, False)],
 2: [(1.0, 342, -1, False)],
 3: [(1.0, 322, -1, False)],
 4: [(1.0, 322, -10, False)],
 5: [(1.0, 322, -10, False)]}
```

Solucionando Sem Aprendizagem Reinforçada

In [7]:

```
env.s = 322
frames = []
epocas = 0
punicoes, recompensa = 0, 0

terminado = False

while not terminado:
    acao = env.action_space.sample() # executa uma acao aleatoria
    estado, recompensa, terminado, info = env.step(acao)

    if recompensa == -10:
        punicoes += 1

    # frames
    frames.append({
        'frame': env.render(mode='ansi'),
        'estado': estado,
        'ação': acao,
        'recompensa': recompensa
    })

    epocas += 1
print('Passos dados (Timesteps): ', epocas)
print('Punições recebidas', punicoes)
```

```
Passos dados (Timesteps):  1304
Punições recebidas 432
```

In [13]:

```
from IPython.display import clear_output
from time import sleep

def print_frames(frames, segundos=0.1):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame['frame'])
        print(f'Passos dados (Timesteps): {i+1}')
        print(f'Estado: {frame["estado"]}')
        print(f'Ação: {frame["ação"]}')
        print(f'Recompensa: {frame["recompensa"]}')
        sleep(segundos)
```

In [14]:

```
print_frames(frames, segundos=0.1)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Passos dados (Timesteps): 1304
Estado: 410
Ação: 5
Recompensa: 20
```

Com Aprendizagem Reinforçada

Q-learning

Essencialmente, o agente vai aprender através das recompensas(positivas e negativas) com um tempo a tomar a melhor decisão para um determinado estado.

- Temos a tabela da recompensa P que é de onde o agente vai aprender, ao tomar uma ação no estado atual e observando a recompensa/punição, atualiza o valor-Q (Q-value).
- O valor-Q para um estado/cenário representa a "qualidade" da ação que ele irá tomar

Os valores-Q são inicializados de forma aleatória, e o agente se expõe ao ambiente, onde recebe diferentes recompensas (positivas e negativas) ao tomar diferentes ações, de forma que os valores-Q são atualizados usando a seguinte fórmula:

$$Q(estado, ação) = (1 - \alpha) \cdot Q(estado, ação) + \alpha\Big(recompensa + \gamma \max_a Q(próximo\ estado, todas\ ações)\Big)$$

Onde:

- α (Alpha) é a taxa de aprendizagem (entre 0 e 1)
- γ (Gamma) é o fator de desconto também (entre 0 e 1), que significa o quanto de importância a gente quer dar para uma recompensa. De forma que 0 faz com que o agente se preocupe apenas com a

recompensa imediata. O ideal é que o agente tome as ações considerando as recompensas do estado atual, e o máximo de recompensa para o próximo estado.

Tabela-Q

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

- A Tabela-Q tem seus valores inicializados como 0, e depois vão sendo atualizados conforme o agente vai tomando ações no ambiente e obtendo o máximo de recompensas

Resumindo

- Inicializa a tabela-Q com zeros
- Começa a explorar o ambiente com ações, seleciona uma de todas as ações possíveis no estado atual (Es_1)

- Vai para o próximo estado (Es_2) como resultado da ação (A_1)
- De todas as possíveis ações no estado (Es_2) seleciona a que possui o maior valor-Q
- Atualiza a tabela-Q usando a equação
- Define o próximo estado como o estado atual
- Se o objetivo é alcançado, termina, senão, repete o processo

In [16]:

```
import numpy as np

tabela_q = np.zeros([env.observation_space.n, env.action_space.n]) #
```

In [17]:

```
tabela_q.shape
```

Out[17]:

```
(500, 6)
```

In [18]:

```
tabela_q
```

Out[18]:

```
array([[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.]])
```

Explorando valores aprendidos

Depois de explorar ações aleatórias, os valores-Q tendem a se divergirem, de forma que o agente vai poder escolher a melhor ação possível para um determinado estado.

Existe um meio termo entre explorar (escolher uma ação aleatória) e usufruir (escolher ações baseadas em valores-Q já aprendidos/treinados). Nós queremos impedir o agente de ficar toda vez fazendo os exatos movimentos, e possivelmente se super-adequando *(overfitting)*. Para evitar isso, usamos mais um parâmetro chamado ϵ "epsilon" para equilibrar essas ações durante o treino do agente.

Ao invés de apenas selecionar o melhor valor-Q, algumas vezes vamos explorar novas ações. Um epsilon pequeno trás mais punições (em média), o que é natural, uma vez que estamos explorando tomando ações aleatórias.

Treinando o Agente

In [29]:

```
%%time
import random

alpha = 0.1
gamma = 0.6
epsilon = 0.1

for i in range(10000):
    estado = env.reset()

    epoca, punicao, recompensa = 0, 0, 0
    terminado = False

    while not terminado:
        if random.uniform(0,1) < epsilon:
            acao = env.action_space.sample() # explorar
        else:
            acao = np.argmax(tabela_q[estado]) # usufruir - escolher acao do melhor

        prox_estado, recompensa, terminado, info = env.step(acao)

        valor_antigo = tabela_q[estado, acao]
        prox_valor_max = np.max(tabela_q[prox_estado])

        # Aplicar a formula
        valor_novo = (1-alpha) * valor_antigo + alpha*(recompensa+gamma*prox_valor_

        tabela_q[estado, acao] = valor_novo

        if recompensa == -10:
            punicao += 1

        estado = prox_estado
        epoca += 1

    clear_output(wait=True)
    print('Episódios',i+1)
print('Finalizado!')
```

```
Episódios 10000
Finalizado!
CPU times: user 1min 27s, sys: 7.88 s, total: 1min 35s
Wall time: 1min 47s
```

In [34]:

```
tabela_q[328]
```

Out[34]:

```
array([-2.34580168, -2.27325308, -2.33622025, -2.32299431, -6.6653213
6,
       -7.53467027])
```

Agora que já treinamos o Agente, não precisamos mais explorar. Vamos apenas selecionar sempre a melhor ação escolhendo o melhor valor-Q.

In [45]:

```
# Avaliar a performance do nosso Agente
epocas, punicoes = 0,0
episodios = 10
frames = []
for _ in range(episodios):
    estado = env.reset()
    terminado = False
    while not terminado:
        acao = np.argmax(tabela_q[estado])
        estado, recompensa, terminado, info = env.step(acao)

        if recompensa == -10:
            punicoes += 1

        frames.append({
            'frame': env.render(mode='ansi'),
            'estado': estado,
            'ação': acao,
            'recompensa': recompensa
        })

        epocas+=1
print(f'Resultados após {episodios} episódios:')
print(f'Épocas {epocas}')
print(f'Punições {punicoes}')
```

```
Resultados após 10 episódios:
Épocas 124
Punições 0
```

In [46]:

```
len(frames)
```

Out[46]:

```
124
```

In [47]:

```
print_frames(frames,0.5)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Passos dados (Timesteps): 124
Estado: 85
Ação: 5
Recompensa: 20
```

Otimizações de Hiperparâmetros

Otimização de Hiperparâmetros

Alpha, gamma e epsilon foram definidos baseados na intuição, mas existem melhores formas de escolher os melhores parâmetros e ter um desempenho melhor.

- α (Alpha) - (Taxa de aprendizagem) = Deveria diminuir com o tempo, para o agente aprender cada vez mais e mais
- γ (Gamma) - Quanto mais próximo você está do objetivo final, maior deveria ser a preferência para a recompensa imediata
- ϵ (Epsilon) - Quanto mais experiência tem o agente, menos precisará explorar. Logo, o epsilon deve diminuir com o tempo.
- Poderíamos aplicar uma pesquisa pelos melhores parâmetros, similar ao GridSearch que vimos para os modelos preditivos.

Fontes (em inglês):

- https://gym.openai.com/ (https://gym.openai.com/)
- https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/ (https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/)