Reinforcement Learning Tutorial

Instructor: Kanan Rahimli

In this project, I trained the Reinforcement Learning Agent to find its way to save the princess on a 5x5 maze. The code is simple and self explanatory but I will point out important parts. The code is written in a way that it can be used for any sized maze regardless of the position of elements:

Size = 25

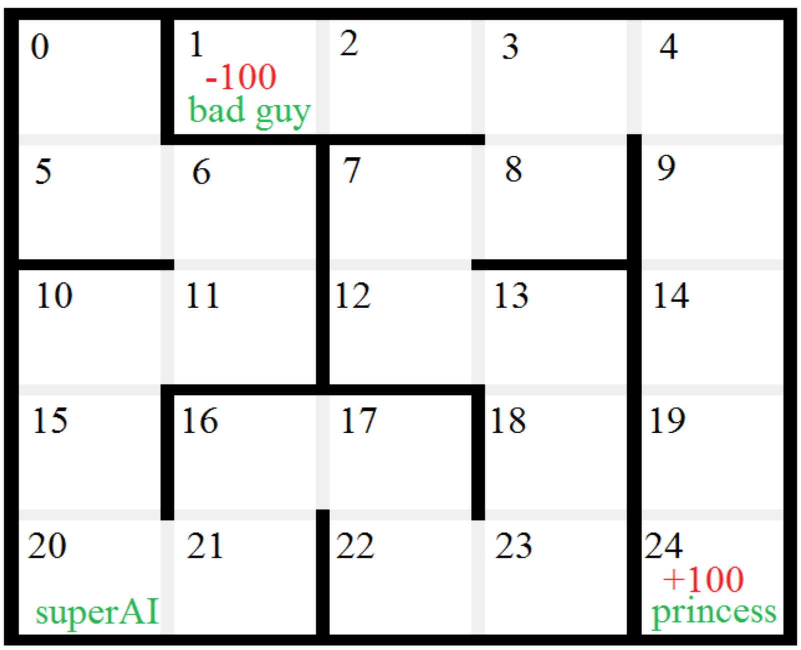
New paths can be added:

allowed_paths = ((0,5), (5,6), (6,11), (10,11), (10,15), (15,20), (20,21),

(16,21), (16,17), (17,22), (22,23), (18,23), (13,18), (12,13),

(7,12), (7,8), (1,2), (2,3), (3,8), (3,4), (4,9), (9,14), (14,19),

(19,24))



For example, on the map there is a wall from 18 to 19 but we can “break” the wall by appending (18, 19) to the list.

We create a reward table filling in with dummy numbers, -1 in this case.

R = np.matrix(np.ones([size,size]))

R *= -1

The agent can 'decide to stay at its current position:

```
for i in range(size):
    R[i,i] = 0
```

and the cost is 0

going back is allowed too:

```
for i in allowed_paths:
    a = i[::-1]
    new_paths.append(a)
allowed_paths = allowed_paths + new_paths
```

and they cost 0 as well

bumping into bad guy costs -100

then we create a Q matrix (memory of the agent) filling with 0, since agent doesn't know anything at first

given a state, we return all the available actions from that state:

```
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row != -1)[1]
    return av_act
```

given all the possible actions, we choose one at random:

```
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_actions_range,1))
    return next_action
```

using the bellman formula, we update the q table, Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]:

```
def update(current_state, action, gamma):


    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]


    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
```

```
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]
    Q[current_state, action] = R[current_state, action] + gamma * max_value
```

train the agent many times:

```
for i in range(size*1000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)
    update(current_state,action,gamma)
```

we run the update function until we find princess, as in we give the agent his meaning of life:

```
while current_state != princess:
    do all the steps...
```

```
The best path to save the princess is:
[11, 10, 15, 20, 21, 16, 17, 22, 23, 18, 13, 12, 7, 8, 3, 4, 9, 14, 19, 24]
>>>
```