What is reinforcement learning?

Reinforcement learning is similar to how we humans learn. When we want to learn how to open a jar, we repeatedly try to open it in many different ways. First clockwise, then anti-clockwise, maybe we'll use a tea towel or our shirts to get a better grip. Eventually, after trial and error, we know how to open a jar.

Reinforcement learning is the same. The machine repeatedly tries one thing after another until it gets it right.

For example, if we have a maze where the machine has to go from one side to the other, the machine will bump and get lost millions of times before it eventually learns how to complete the maze and even the fastest way through the maze.

What is an agent?

The agent is the little person that explores the maze! We traditionally call them agents, because we tell them to accomplish something and they do it. And also, universities prefer agents over "tiny computerised humans".

What is a rewards table?

The way agents learn is that they store all the information they gather in a table. Specifically, the information gained from an agent is:

- Where did the agent start?
- What is the new location of the agent?
- What is the reward of moving this way?

Take, for instance, the agent is at the start at (0, 0). If the agent moves right, that might give a reward of 0.2. If the agent moves down, it might give a reward of 0.5. The agent now knows the best move is to move down.

The agent has to write these things down. As humans, we also use memory to remember things. "This door opens inwards", "to unlock the front door you have to push the lockdown and the door handle at the same time" and so on. The agent has a memory, just like us.

Task 1

We're going to change how *fast* the agent learns, and we're going to explore what this will result in.

The learning rate is the speed at which the agent learns. Set it to 1.0, and the agent learns extremely fast. Set it to 0, and the agent doesn't learn at all.

The learning rate can only be between 0 and 1, think of increments like 0.25, 0.7831 and so on.

However, simply increasing it to 1 does not mean the agent finds the best route through the maze. Because the agent learns so fast, it might be rushing through the maze missing all the import steps to finding the fastest way through it.

But setting it too low such as 0.01 means the agent is very slow, and it will take a longer time for the program to complete.

Open the file `Qagent.py`, and find this part in the file (it's on line 13):

```
"""
CHANGE VARIABLES BELOW THIS LINE
"""
# How fast can the agent learn?
self.learningRate = 0.25
```

1. Set the learning rate to 0.5.

Now, run the file `maze.py` and observe the output. **Note: every time we want to run the agent, we have to run maze.py**

Look at the path the agent took, and the rewards table. How big is the path? How small is the rewards table?

Play around with the learning rate. Set it to any number between 0 and 1, and see for yourself how changing the pace of learning the agent's outputs are different.

Task 2

We're going to play around with the `self.explore` variable now.

The explore variable is a probability between 0 and 1 (where 0.15 is 15%).

In reinforcement learning, we have a tradeoff of exploiting what we know or trying something different.

Take, for instance, buttering bread.

If we butter bread like a circle, going clockwise around the bread it's not very efficient. But if we never explore, we would never find the most efficient way to butter the bread.

If the agent only does what it knows, it will never find the most efficient route through the maze.

The exploratory variable means that every move the agent takes, it has a 15% chance to randomly select a move, regardless of whether or not it understands how rewarding the move is.

Find the exploratory variable at line 37 of Qagent.

```
# The probability that the agent will explore on that round, instead of
exploiting the rewards table
self.explore = 0.15
```

1. Set the exploratory percentage to 100 (`self.explore = 1`) See how the agent has lost its memory? It has a memory, but it doesn't use its memories! Every move it makes is random.

What if we set explore to 0? Will the agent find the best route through the maze or not?

Task 3

Now we're going to understand the randomness aspect of exploratory. Because the agent explores, that means every time the agent attempts the maze will be different from the last time. Run your programs and have a look at the person next to you. See how they're different routes? Different rewards? One of your agents is smarter than the other!

But, you might see a problem. There is only 1 route which is the fastest way through the maze. But if there is 1 route and every agent is slightly different because of the random explorations, how does it find the best route?

The answer is using epochs.

Epoch is a fancy word for "time". Specifically, we might tell the agent:

> "Do this maze 10 billion times" If we make 2 agents do the maze 2 or 3 times, their answers will be completely different. But if we make the agents do the maze 10 billion times, they will *converge* on the correct answer - the fastest route through the maze!

Set the epochs variable to a high number, such as `10000`. However! Your screen might go crazy, so try not to read every single line that comes out of the program.

You can find epochs on line 34 of Qagent.

```
# How many times will the agent complete the maze before it stops?
self.max_epochs = 1
```

Note: You will need to set max_epochs back to 1 for the rest of this tutorial.

Task 4

Let's talk about gamma!

Gamma is the value of future reward. If Gamma is set to 1, the agent sees reaching the goal in 10 moves and reaching the goal in 1 move as the same value.

If gamma is set to 0, the agent values direct moves (what it's doing on the next turn) more than it does in the future.

When we humans try to solve a maze, we know where the exit is and we try to tend towards the exit. If the agent's gamma was 0, the agent wouldn't tend towards the exit. The agent would simply try to collect as much reward as possible until it accidentally stumbles upon the exit.

Find gamma on line 31 of Qagent:

```
# How much the agent expects to gain from future value.
self.gamma = 0.5
```

1. Set the gamma rate to 1.
2. Set the gamma rate to 0.

See how they differ?

Task 5

Now, we're going to work on punishing the agent. To prevent the agent from wandering around in circles, the agent takes a punishment every time it moves.

If the punishment is too much, the agent gives up and we get a new agent to try (but with the same rewards table as the last agent).

Increasing the punishment means the agent gives up faster, but also means the agent is more encouraged to find the goal faster.

Too low of a punishment and the agent will think "why bother?" and will reach the goal in a much slower method.

Find this on line 43 of Qagent:

```
# The agent's penalty for moving. Prevents the agent from running around in
circles
self.penaltyMoving = -0.05
```

1. Set the penalty to -1.
2. Set the penalty to 0.

Observe the outputs.

Task 6

This is a more advanced task, so be prepared!

The maze the agent uses is located in `maze.py` (line 12), specifically this is it:

```
self.maze = np.array([
    [ 1,  0,  1,  1,  1,  1,  1,  1,  1,  1],
    [ 1,  1,  1,  1,  1,  0,  1,  1,  1,  1],
    [ 1,  1,  1,  1,  1,  0,  1,  1,  1,  1],
```

```
    [ 0,  0,  1,  0,  0,  1,  0,  1,  1,  1],
    [ 1,  1,  0,  1,  0,  1,  0,  0,  0,  1],
    [ 1,  1,  0,  1,  0,  1,  1,  1,  1,  1],
    [ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
    [ 1,  1,  1,  1,  1,  1,  0,  0,  0,  0],
    [ 1,  0,  0,  0,  0,  0,  1,  1,  1,  1],
    [ 1,  1,  1,  1,  1,  1,  1,  0,  1,  1]
])
```

The 1 represents openness, a straight corridor for the agent. The 0's represent a wall, something the agent can not pass.

What if we make the goal impossible to reach? The goal is in the bottom right-hand corner.

1. Change some of the 1's to 0's (walls) to prevent the agent from reaching the goal. What will happen then?

Play around with changing the layout of the maze and see how the agent reacts.

Task 7

Now we're going to change what moves the agent knows.

On line 28 of `Qagent.py` the agent defines the moveeset as:

```
self.possibleActions = {"Up": (1, 0), "Down": (-1, 0), "Left": (0, -1),
"Right": (0, 1)}
```

The agent can move up, down, left, or right.

It does this using vector addition. On a 2 dimensional map, adding (1, 0) to our current coordinates (15, 15) means we would go up 1 square to (16, 15).

1. Try to remove some moves. You can do this by deleting the entire dictionary entry. For example, to reduce Up do this:

```
self.possibleActions = {"Down": (-1, 0), "Left": (0, -1), "Right": (0, 1)}
```

Don't forget to remove the comma!

Task 8

Now we're going to change the agents start location. Currently, it starts at the top left (0, 0). But it can start anywhere!

The way the coordinates system works is (row, column). "Along the corridor, up the stairs".

Find the code on line 26 of `maze.py`:

```
self.agentLocation = (0, 0)
```

1. Try setting the agent location to (5, 5) and seeing what the agent does! Does it walk to the top left again? Or go straight to the goal.

Task 9

Change the goal.

The goal variable is located at line 21 of `Qagent.py`.

```
self.goal = self.maze.getSizeOfMaze()
```

1. Change the goal to a location somewhere on the maze. For example, (15, 15) like so:

```
self.goal = (15, 15)
```

2. If you set the goal and start location to the same place, what happens?

Task 10

This is an advanced task.

1. Change how the agent learns

Some things you may want to do:

- Remove the penalty completely (by setting it to 0.0)
- Remove the agent giving up when it is punished too much
- Play with the learning rate
- Play with the gamma rate

Try to find the most optimal settings which result in the most optimal agent. Often, we would program a computer to automatically find the most optimal settings. But it is possible to deduct and use logic to work out roughly what the most optimal settings are.

Google things you don't understand such as "learning rate" or "gamma q learning" to get the answers you're looking for.