Open in Colab

THE AGENT WHO AVOIDED THE ORDINARY

In this lab, you will implement the simplest agent that learns to predict the outcome of its actions and tries another action when it gets bored.

Learning objective

Upon completing this lab, you will be able to implement artificial agents based on the 'conceptual inversion of the interaction cycle.' In this framwork, the agent starts by taking action and then receives a sensory signal which is an outcome of action. This contrasts with traditional AI agents, which first perceive their environment before deciding how to act.

Setup

Define the Agent class

In [10]:
```python
import random


class Agent:
    def __init__(self):
        """ Creating our agent """
        self._action = None
        self._predicted_outcome = None

    def action(self, _outcome):
        """ tracing the previous cycle """
        if self._action is not None:
            print(f"Action: {self._action}, Prediction: {self._predicted_outcome
                  f"Satisfaction: {self._predicted_outcome == _outcome}")

        """ Computing the next action to enact """
        # TODO: Implement the agent's decision mechanism
        self._action = 0
        # TODO: Implement the agent's anticipation mechanism
        self._predicted_outcome = 0
        return self._action
```

Environment1 class

In [11]:
```python
class Environment1:
    """ In Environment 1, action 0 yields outcome 0, action 1 yields outcome 1 "
    def outcome(self, _action):
        # return int(input("entre 0 1 ou 2"))
        if _action == 0:
            return 0
        else:
            return 1
```

Environment2 class

In [12]:
```
class Environment2:
    """ In Environment 2, action 0 yields outcome 1, action 1 yields outcome 0 "
    def outcome(self, _action):
        if _action == 0:
            return 1
        else:
            return 0
```

Instantiate the agent

In [13]:
```
a = Agent()
```

Instantiate the environment

In [14]:
```
e = Environment1()
```

Test run the simulation

In [15]:
```
outcome = 0
for i in range(10):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
```

Observe that, on each interaction cycle, the agent correctly predicts the outcomes. The agent's satisfaction is True because its predictions are correct.

PRELIMINARY EXERCISE

Run the agent in Environment2. Observe that its satisfaction becomes False. This agent is not satisfied in Environment2!

Now you see the goal of this assignment: design an agent that learns to be satisfied when it is run either in Environment1 or in Environment2.

ASSIGNMENT

Implement Agent1 that:

- learns to predict the outcome of its actions
- chooses a different action when its predictions have been correct for 4 times in a row

The agent can choose two possible actions `0` or `1`, and can recieve two possible outcomes: `0` or `1`.

It computes the prediction on the assumption that the same action always yeilds the same outcome in a given environment. You must thus implement a memory of the obtained outcomes for each action.

Create your own agent by overriding the class Agent

Create an agent that learns to correctly predict the outcome of its actions in both Environment1 and Environment2.

You may add any class attribute or method you deem useful.

In [16]:
```
class Agent1(Agent):
    def __init__(self):
        super().__init__()
        self.action_outcome = {0: None, 1: None}
        self.nb_corrects_in_a_row = 0
        self._action = None
        self._predicted_outcome = None

    # TODO override the method action(self, _outcome)
    def action(self, _outcome):
        """ On affiche les résultats précédents """
        if self._action is not None:
            print(f"Action: {self._action},"
                  f" Prediction: {self._predicted_outcome},"
                  f" Outcome: {_outcome}, "
                  f"Satisfaction: {self._predicted_outcome == _outcome}")

        """ On mémorise l'outcome de l'action précédente """
        self.action_outcome[self._action] = _outcome

        """ Si la prédiction est correcte, on incrémente le compteur de corrects
        if self._predicted_outcome == _outcome:
            self.nb_corrects_in_a_row += 1
        else:
            self.nb_corrects_in_a_row = 0
        """ On choisit la prochaine action """
        # Si on ne sait pas quelle action choisir, on met 0 par défaut
        if self._action is None:
            self._action = 0
        # Si on a eu 4 corrects d'affilée, on change d'action
        if self.nb_corrects_in_a_row == 4:
            # action opposée (action = 1 - action) [1 - 0 => 1, 1 - 1 => 0]
            self._action = 1 - self._action
            self.nb_corrects_in_a_row = 0

        """ Selon l'action choisie, on prédit l'outcome si on peut """
        self._predicted_outcome = self.action_outcome[self._action]
        # Si on ne sait pas, on met une valeur par défaut
        if self._predicted_outcome is None:
            self._predicted_outcome = 0

        return self._action
```

Test your agent in Environment1

In [17]:
```
a = Agent1()
e = Environment1()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 1, Satisfaction: False
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 0, Outcome: 0, Satisfaction: True
```

Test your agent in Environment2

In [18]:
```
a = Agent1()
e = Environment2()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 1, Satisfaction: False
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 1, Prediction: 0, Outcome: 0, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
Action: 0, Prediction: 1, Outcome: 1, Satisfaction: True
```

Report

Explain what you programmed and what results you observed. Export this document as PDF including your code, the traces you obtained, and your explanations below (no more than a few paragraphs):

Pour l'Agent1, nous avons développé un agent capable de choisir des actions et de prédire leur résultat en fonction de l'environnement dans lequel il se trouve. Initialement, l'agent ne sait pas quelles sont les sorties associées à chaque action, ce qui signifie qu'il peut faire des erreurs lors de ses premières tentatives (au plus une erreur par action).

Cependant, une fois qu'il effectue une action et observe l'issue, l'agent mémorise cette association entre l'action et l'issue. Ainsi, s'il effectue à nouveau la même action, il pourra prédire correctement l'issue en se basant sur ses expériences passées.

Afin que l'agent ne s'ennuye pas, nous avons mis en place un compteur qui suit le nombre de prédictions correctes consécutives. Lorsque l'agent fait quatre prédictions correctes d'affilée, il change automatiquement d'action. Cela permet d'éviter que l'agent ne répète indéfiniment les mêmes actions.

La trace de l'agent dans l'environnement 1

No description has been provided for this image

La trace de l'agent dans l'environnement 2

No description has been provided for this image