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

Open in Colab

THE AGENT WHO THRIVED ON GOOD VIBES

Learning objectives

Upon completing this lab, you will be able to implement agents driven by a type of intrinsic motivation called 'interactional motivation.' This refers to the drive to engage in sensorimotor interactions that have a positive valence while avoiding those that have a negative valence.

Setup

Define the Agent class

In [40]:
```
class Agent:
    def __init__(self, _valences):
        """ Creating our agent """
        self._valences = _valences
        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"Prediction: {self._predicted_outcome == _outcome}, Valence:

        """ 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 [41]:
```
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 [42]:
```
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
```

Define the valence of interactions

In [43]:
```
valences = [[-1, 1],
            [1, -1]]
```

The valence table specifies the valence of each interaction. An interaction is a tuple (action, outcome):

	outcome 0	outcome 1
action 0	-1	1
action 1	1	-1

Instantiate the agent

In [44]:
```
a = Agent(valences)
```

Instantiate the environment

In [45]:
```
e = Environment1()
```

Test run the simulation

In [46]:
```
outcome = 0
for i in range(10):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
```

Observe that, on each interaction cycle, the agent is mildly satisfied. On one hand, the agent made correct predictions, on the other hand, it experienced negative valence.

PRELIMINARY EXERCISE

Execute the agent in Environment2. Observed that it obtains a positive valence.

Modify the valence table to give a positive valence when the agent selects action `0` and obtains outcome `0`. Observe that this agent obtains a positive valence in Environment1.

ASSIGNMENT

Implement Agent2 that selects actions that, it predicts, will result in an interaction that have a positive valence.

Only when the agent gets bored does it select an action which it predicts to result in an interaction that have a negative valence.

In the trace, you should see that the agent learns to obtain a positive valence during several interaction cycles. When the agent gest bored, it occasionnaly selects an action that may result in a negative valence.

Create Agent2 by overriding the class Agent

In [47]:
```
class Agent2(Agent):
    def __init__(self, _valences):
        super().__init__(_valences)
        self.action_outcome = {0: None, 1: None}
        self.nb_corrects_in_a_row = 0
        self._action = None
        self._predicted_outcome = None
        self._valences = _valences


    # 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}, Prediction: {self._predicted_outcome
                  f"Prediction: {self._predicted_outcome == _outcome}, Valence:

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

        # Si on ne sait pas quelle action choisir, on met 0 par défaut
        if self._action is None:
            self._action = 0

        """ Si la prédiction est correcte, on incrémente le compteur de corrects
        if self._predicted_outcome == _outcome or self._valences[self._action][_
            self.nb_corrects_in_a_row += 1
```

```
        else:
            self.nb_corrects_in_a_row = 0

        """ On choisit la prochaine action """
        # Si on a eu 4 corrects d'affilée, on change d'action
        if self.nb_corrects_in_a_row == 4 or self._valences[self._action][_outco
            # 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 Agent2 in Environment1

In [48]:

```
a = Agent2(valences)
e = Environment1()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 1, Prediction: 0, Outcome: 1, Prediction: False, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
```

Test your Agent2 in Environment2

In [49]:

```
a = Agent2(valences)
e = Environment2()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

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

Test your agent with a different valence table

Note that, depending on the valence that you define, it may be impossible for the agent to obtain a positive valence in some environments.

In [50]:

```python
# Choose different valences
valences = [[1, -1],
            [1, -1]]
# Run the agent
a = Agent2(valences)
e = Environment2()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: -1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: -1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1
```

Report

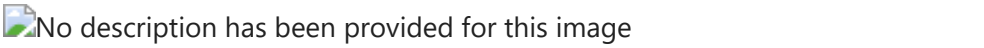
Explain what you programmed and what results you observed. Export this document as PDF including your code, the traces you obtained, and your explanations below (no more than a few paragraphs):

Pour l'Agent2, nous avons développé un agent qui en plus de l'Agent1 prend en compte des interaction. Une interaction est un tuple (action, outcome) qui a une certaine valeur. Ici, les valeurs des interactions sont soit négative (-1), soit positive (1). L'agent2 essaie toujours de prédire la sortie de l'action qu'il a choisit mais regarde et prends en compte la valence l'interaction qui en découle. L'agent2 va préférer une valence positive à une valence négative. De ce fait, si il reçoit une valence négative il va changer d'action pour choisir une action qui lui donne un valence positive, tout en faisant attention que l'agent ne s'ennuie pas et que si il commence à s'ennuyer, alors il va changer d'action même si cela implique d'avoir une valence négative.

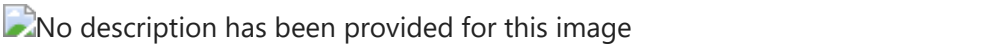
Tableau de valence utilisé pour les 2 premières traces:

	outcome 0	**outcome 1**
action 0	-1	1
action 1	1	-1

Dans l'image ci-dessous l'agent évoluant dans l'environnemen1 ne reçoit que des valences négatives. Cela est dû au tableau de valence qui fait que peut importe l'action que choisit l'agent il recevra toujours une valence négative. Par conséquent il va à chaque fois changer l'action


No description has been provided for this image

A l'inverse, toujours à cause du tableau de valence choisit, dans l'environnemnt2, l'agent ne va recevoir que des valences positives ce qui implique qu'il ne va changer d'action uniquelent lorsqu'il va s'ennuyer


No description has been provided for this image

Pour avoir quelque chose de moins binaire (recevoir que des valences positives ou que négatives), nous avons défini un nouveau tableau de valencesqui en fonction de l'action choisi revoit soit une valence positive soit une valence négative.

Tableau de valence choisit:

	outcome 0	**outcome 1**
action 0	1	-1
action 1	1	-1

Ci-dessous la trace de l'agent2 dans l'environnement2 avec le nouveau tableau de valence. Avec ce tableau de valence et dans cet environnement, lorque l'agent choisit l'action 1 il valence recevoir une valence positive et une valence négative pour l'action 0. L'agent va donc effectuer l'action 1 jusqu'à ce qu'il s'ennuie pouis l'action 0 une fois et revenir à l'action 1.

 No description has been provided for this image

Open in Colab

THE AGENT WHO TAMED THE TURTLE

Learning objectives

Upon completing this lab, you will be able to assign appropriate valences to interactions, enabling a developmental agent to exhibit exploratory behavior in a simulated environment.

Setup

Import the turtle environment

In [36]:
```
!pip3.10 install ColabTurtle
from ColabTurtle.Turtle import *
```

```
Requirement already satisfied: ColabTurtle in /usr/local/lib/python3.10/site-pack
ages (2.1.0)
```

Define the Agent class

In [59]:
```
class Agent:
    def __init__(self, _valences):
        """ Creating our agent """
        self._valences = _valences
        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"Prediction: {self._predicted_outcome == _outcome}, Valence:

        """ 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
```

Define the tutle environment class

You don't need to worry about the code of the ColabTurtleEnvironment below.

Just know that this environment:

- interprets the agent's actions as follows `0`: move forward, `1`: turn left, `2`: turn right.
- returns outcome `1` when the turtle bumps into the border of the window, and `0` otherwise.

In [60]:
```
# @title Initialize the turtle environment

BORDER_WIDTH = 20

class ColabTurtleEnvironment:

    def __init__(self):
        """ Creating the Turtle window """
        bgcolor("lightGray")
        penup()
        goto(window_width() / 2, window_height()/2)
        face(0)
        pendown()
        color("green")

    def outcome(self, action):
        """ Enacting an action and returning the outcome """
        _outcome = 0
        for i in range(10):
            # _outcome = 0
            if action == 0:
                # move forward
                forward(10)
            elif action == 1:
                # rotate left
                left(4)
                forward(2)
            elif action == 2:
                # rotate right
                right(4)
                forward(2)

            # Bump on screen edge and return outcome 1
            if xcor() < BORDER_WIDTH:
                goto(BORDER_WIDTH, ycor())
                _outcome = 1
            if xcor() > window_width() - BORDER_WIDTH:
                goto(window_width() - BORDER_WIDTH, ycor())
                _outcome = 1
            if ycor() < BORDER_WIDTH:
                goto(xcor(), BORDER_WIDTH)
                _outcome = 1
            if ycor() > window_height() - BORDER_WIDTH:
                goto(xcor(), window_height() -BORDER_WIDTH)
                _outcome = 1

            # Change color
            if _outcome == 0:
                color("green")
            else:
                # Finit l'interaction
                color("red")
                # if action == 0:
```

```
                #     break
                if action == 1:
                    for j in range(10):
                        left(4)
                elif action == 2:
                    for j in range(10):
                        right(4)
                break

        return _outcome
```

Define the valence of interactions

In [61]:
```
valences = [[1, -1],
            [-1, -1],
            [-1, -1]]
```

The valence table specifies the valence of each interaction. An interaction is a tuple (action, outcome):

	0 Not bump	1 Bump
0 Forward	1	-1
1 Left	-1	-1
2 Right	-1	-1

Instantiate the agent

In [62]:
```
a = Agent(valences)
```

Run the simulation

In [63]:
```
# @title Run the simulation

initializeTurtle()

# Parameterize the rendering
bgcolor("lightGray")
penup()
goto(window_width() / 2, window_height()/2)
face(0)
pendown()
color("green")
speed(10)

e = ColabTurtleEnvironment()
print("Outcome:")
outcome = 0
for i in range(10):
    action = a.action(outcome)
    outcome = e.outcome(action)
    print("Outcome:")
```

Observe the turtle moving in a straigt line until it bumps into the border of the window

PRELIMINARY EXERCISE

Copy Agent2 that you designed in your previous assignment to this notebook.

Observe how your Agent2 behaves in this environment

ASSIGNMENT

Implement Agent3 by modifying your previous Agent2 such that it can select 3 possible actions: `0`, `1`, or `2`.

Choose the valences of interactions so that the agent does not remain stuck in a corner of the environment.

Create Agent3 by overriding the class Agent or your previous class Agent2

In [51]:
```
class Agent3(Agent):
    def __init__(self, _valences):
        super().__init__(_valences)
        self.action_outcome = {0: None, 1: None, 2: None}
        self.nb_corrects_in_a_row = 0
        self._action = None
        self._predicted_outcome = None
        self._valences = _valences


    # 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}, Prediction: {self._predicted_outcome
                  f"Prediction: {self._predicted_outcome == _outcome}, Valence:

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

        # Si on ne sait pas quelle action choisir, on met 0 par défaut
        if self._action is None:
            self._action = 0

        """ Si la prédiction est correcte, on incrémente le compteur de corrects
        if self._predicted_outcome == _outcome or self._valences[self._action][_
            self.nb_corrects_in_a_row += 1
        else:
            self.nb_corrects_in_a_row = 0

        """ On choisit la prochaine action """
        # Si on a eu 4 corrects d'affilée, on change d'action
        if self.nb_corrects_in_a_row == 4 or self._valences[self._action][_outco
            # action opposée (action = 1 - action) [1 - 0 => 1, 1 - 1 => 0]
            self._action = (self._action + 1) % 3
            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
```

Choose the valence table

Replace the `valences` table by your choice in the code below

In [52]:
```
valences = [[1, -1],
            [1, -1],
            [0, 1]]
```

Test your agent in the TurtleEnvironment

In [53]:
```
initializeTurtle()

# Parameterize the rendering
bgcolor("lightGray")
penup()
goto(window_width() / 2, window_height()/2)
face(0)
pendown()
color("green")
speed(10)

a = Agent3(valences)
```

```
e = ColabTurtleEnvironment()

outcome = 0
for i in range(100):
    action = a.action(outcome)
    outcome = e.outcome(action)
```


Improve your agent's code

If your agent gets stuck against a border or in a corner, modify the valences or the code. Try different ways to handle boredome or to select random actions. In the next lab, you will see how to design an agent that can adapt to the context.

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):

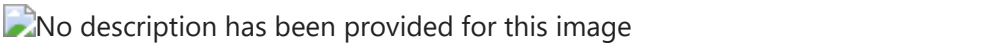
Pour coder notre Agent3 nous avons repris le code pour l'Agent2 et avons introduit une nouvelle action possible (action 2) ainsi que la manière de changer d'action lorsque l'Agent reçoit une valence négative ou s'ennuie. Pour passer d'une action à l'autre nous prennons l'action suivante par exemple si il executait l'action0 il effectura l'action1, pour l'action1 il passera à l'action2 et pour l'action2 reviendra à l'ction0. Nous avons aussi ajusté la table de valence qui est maintenent :

	0 Not bump	1 Bump
0 Forward	1	-1
1 Left	1	-1

	0 Not bump	1 Bump
2 Right	0	1

Le fait de mettre une valence posistive à l'action 2 lorsqu'il but permet à l'agent d'avoir une porte de sortie lorsqu'il but contre un mur et par conséquent de ne pas resté concé dans un coin ou sur un côté du terrain.

Ci-dessous le resultat de l'agent sur 100 epoque avec la trace pour les 21 premières époques:

 No description has been provided for this image

On remarque dès que l'agent se heurte à un bord du terrain, il s'en éloigne assez rapidement en ne reste jamis concé quelque part. Ce qui répond au critères que l'agent 3 doit respecter.

Open in Colab

THE AGENT WHO SHIFTED WITH THE CONTEXT

Learning objectives

Upon completing this lab, you will be able to implement a developmental agent driven by interactional motivation that adapts its next action based on the context of the previously enacted interaction.

Define the Interaction class

Let's define an Interaction class that will be useful to intitialize the agent and to memorize the context

In [104…

```
class Interaction:
    """An interaction is a tuple (action, outcome) with a valence"""
    def __init__(self, action, outcome, valence):
        self.action = action
        self.outcome = outcome
        self.valence = valence

    def key(self):
        """ The key to find this interaction in the dictinary is the string '<ac
        return f"{self.action}{self.outcome}"

    def __str__(self):
        """ Print interaction in the form '<action><outcome:<valence>' for debug
        return f"{self.action}->{self.outcome}:{self.valence}"

    def __eq__(self, other):
        """ Interactions are equal if they have the same key """
        return self.key() == other.key()
```

Define the Agent class

The agent is initialized with the list of interactions

The previous action and the predicted outcome are memorized in the attribute `_intended_interaction`.

In [105…

```
class Agent:
    """Creating our agent"""
    def __init__(self, _interactions):
        """ Initialize the dictionary of interactions"""
        self._interactions = {interaction.key(): interaction for interaction in
```

```
        self._intended_interaction = self._interactions["00"]

    def action(self, _outcome):
        """ Tracing the previous cycle """
        previous_interaction = self._interactions[f"{self._intended_interaction.
        print(f"Action: {self._intended_interaction.action}, Prediction: {self._
              f"Prediction: {self._intended_interaction.outcome == _outcome}, Va

        """ Computing the next interaction to try to enact """
        # TODO: Implement the agent's decision mechanism
        intended_action = 0
        # TODO: Implement the agent's prediction mechanism
        intended_outcome = 0
        # Memorize the intended interaction
        self._intended_interaction = self._interactions[f"{intended_action}{inte
        return intended_action
```

Environment1 class

In [106…

```
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 [107…

```
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
```

Environment3 class

Environment 3 yields outcome 1 only when the agent alternates actions 0 and 1

In [108…

```
class Environment3:
    """ Environment 3 yields outcome 1 only when the agent alternates actions 0
    def __init__(self):
        """ Initializing Environment3 """
        self.previous_action = 0

    def outcome(self, _action):
        if _action == self.previous_action:
            _outcome = 0
        else:
            _outcome = 1
```

```
        self.previous_action = _action
        return _outcome
```

Initialize the interactions

In [109…

```
interactions = [
    Interaction(0,0,-1),
    Interaction(0,1,1),
    Interaction(1,0,-1),
    Interaction(1,1,1),
    Interaction(2,0,-1),
    Interaction(2,1,1)
]
```

Interactions are initialized with their action, their outcome, and their valence:

	outcome 0	outcome 1
action 0	-1	1
action 1	-1	1
action 2	-1	1

Instantiate the agent

In [110…

```
a = Agent(interactions)
```

Instantiate the environment

In [111…

```
e = Environment3()
```

Test run the simulation

In [112…

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

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
```

Observe that in Envirnment3, the agent obtains only negative valences. To obtain a positive valence, it must select a different action on each interaction cycle.

PRELIMINARY EXERCISE

Execute the agent in Environment1. Observe that it obtains a negative valence.

Execute the agent in Environment2. Observe that it obtains a positive valence.

Now you see the goal of this assignement: design an agent that can obtain positive valences when it is run either in Environment1 or in Environment2 or in Environment3.

ASSIGNMENT

Implement Agent4 that obtains positive valences in either Environment 1, 2, or 3.

Agent4 must be able to predict the outcome resulting from its next action depending on the context of the previous interaction. Based on this prediction, it must select the action that will yield the heighest valence.

To do so, at the end of cycle `t`, Agent4 must memorize `interaction_t = (action_t, outcome_t)` that was just enacted. The agent must choose the next `interaction_t+1` based on `interaction_t` (the context). For each possible `action_t+1`, the agent must predict the expected `outcome_t+1`. Based on this prediction, it must select the action that yields the highest `valence_t+1`.

Create Agent4 by overriding the class Agent

You may add any attribute and method you deem usefull to the class Agent4

In [113…

```
class Agent4(Agent):
    def __init__(self, _interactions):
        super().__init__(_interactions)
        self._memory = {}
        self._possible_actions = [0, 1, 2]
        self._nb_corrects_in_a_row = 0
        self._context = []

        for possible_action in self._possible_actions:
            for possible_outcome in [0, 1]:
                self._memory[f"{possible_action}{possible_outcome}"] = list()

    # TODO override the method action(self, _outcome)
    def action(self, _outcome):
        """ Affichage de l'interaction précédente """
        previous_interaction = self._interactions[f"{self._intended_interaction.
        print(f"Action: {self._intended_interaction.action}, Prediction: {self._
              f"Prediction: {self._intended_interaction.outcome == _outcome}, Va

        """ Enregistrement de la suite de 2 interactions précédentes """
        self._context.append(previous_interaction)
        if len(self._context) == 2:
            i1, i2 = self._context
```

```
            if i2 not in self._memory[f"{i1.action}{i1.outcome}"]:
                self._memory[f"{i1.action}{i1.outcome}"].append(i2)

            self._context = []
            self._context.append(i2)

        intended_action = 0
        intended_outcome = 0

        if self._intended_interaction.outcome == previous_interaction.outcome:
            self._nb_corrects_in_a_row += 1
        else:
            self._nb_corrects_in_a_row = 0

        """ Choix de l'intéraction à réaliser avec la plus grande valence """
        next_moves = self._memory[previous_interaction.key()]
        if next_moves:
            best_move = max(next_moves, key=lambda x: x.valence)
            intended_action = best_move.action
            intended_outcome = best_move.outcome
            intended_valence = best_move.valence

            if self._intended_interaction.outcome != previous_interaction.outcom
                intended_action, intended_outcome = self.change_action_outcome(i
            """ if self._nb_corrects_in_a_row > 2:
                intended_action, intended_outcome = self.change_action_outcome(i
                _nb_corrects_in_a_row = 0"""

        # Memorize the intended interaction
        self._intended_interaction = self._interactions[f"{intended_action}{inte
        return intended_action

    def print_memory(self):
        for i1, i2 in self._memory.items():
            print(f"{i1} -> [", end="")
            for v in i2:
                print(f"{v}", end=", ")
            print("]")

    def change_action_outcome(self, intended_action, intended_outcome):
        return (intended_action + 1) % 3, (intended_outcome + 1) % 2
```

Test your Agent4 in Environment1

In [114…

```
a = Agent4(interactions)
e = Environment1()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
```

Test your Agent4 in Environment2

In [115…

```
a = Agent4(interactions)
e = Environment2()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 1, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
```

Test your Agent4 in Environment3

In [116…

```
a = Agent4(interactions)
e = Environment3()
outcome = 0
for i in range(20):
```

```
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 0, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
Action: 1, Prediction: 1, Outcome: 1, Prediction: True, Valence: 1)
```

Test your Agent4 with interactions that have other valences

Replace the valences of interactions with your choice in the code below

In [117…

```
# Choose different valence of interactions
interactions = [
    Interaction(0,0,1),
    Interaction(0,1,0),
    Interaction(1,0,-1),
    Interaction(1,1,1),
    Interaction(2,0,-1),
    Interaction(2,1,1)
]
# Run the agent
a = Agent4(interactions)
e = Environment3()
outcome = 0
for i in range(20):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
```

Test your agent in the Turtle environment

In [118…

```
# @title Install the turtle environment
!pip3 install ColabTurtle
from ColabTurtle.Turtle import *
```

```
Requirement already satisfied: ColabTurtle in c:\users\nassm\appdata\local\progra
ms\python\python312\lib\site-packages (2.1.0)
```

In [119…

```
# @title Initialize the turtle environment

BORDER_WIDTH = 20

class ColabTurtleEnvironment:

    def __init__(self):
        """ Creating the Turtle window """
        bgcolor("lightGray")
        penup()
        goto(window_width() / 2, window_height()/2)
        face(0)
        pendown()
        color("green")

    def outcome(self, action):
        """ Enacting an action and returning the outcome """
        _outcome = 0
        for i in range(10):
            # _outcome = 0
            if action == 0:
                # move forward
                forward(10)
            elif action == 1:
                # rotate left
                left(4)
                forward(2)
            elif action == 2:
                # rotate right
```

```
                right(4)
                forward(2)

            # Bump on screen edge and return outcome 1
            if xcor() < BORDER_WIDTH:
                goto(BORDER_WIDTH, ycor())
                _outcome = 1
            if xcor() > window_width() - BORDER_WIDTH:
                goto(window_width() - BORDER_WIDTH, ycor())
                _outcome = 1
            if ycor() < BORDER_WIDTH:
                goto(xcor(), BORDER_WIDTH)
                _outcome = 1
            if ycor() > window_height() - BORDER_WIDTH:
                goto(xcor(), window_height() -BORDER_WIDTH)
                _outcome = 1

            # Change color
            if _outcome == 0:
                color("green")
            else:
                # Finit l'interaction
                color("red")
                # if action == 0:
                #     break
                if action == 1:
                    for j in range(10):
                        left(4)
                elif action == 2:
                    for j in range(10):
                        right(4)
                break

        return _outcome
```

In [120…

```
# @title Run the turtle environment
initializeTurtle()

# Parameterize the rendering
bgcolor("lightGray")
penup()
goto(window_width() / 2, window_height()/2)
face(0)
pendown()
color("green")
speed(13)

a = Agent4(interactions)
e = ColabTurtleEnvironment()

outcome = 0
for i in range(50):
    action = a.action(outcome)
    outcome = e.outcome(action)
```

```
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 1, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 1, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 1, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 0, Prediction: True, Valence: -1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 0, Prediction: True, Valence: 1)
Action: 0, Prediction: 0, Outcome: 1, Prediction: False, Valence: 0)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
Action: 2, Prediction: 0, Outcome: 1, Prediction: False, Valence: 1)
```

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):

Classe `Agent4`

L'agent `Agent4` utilise les interactions passées pour prédire et choisir des actions qui maximisent la valence des résultats.

Attributs et Initialisation

- **_memory** : Dictionnaire enregistrant des séquences d'interactions précédentes (action + résultat).
- **_possible_actions** : Actions possibles `[0, 1, 2]`.
- **_nb_corrects_in_a_row** : Compteur de prédictions correctes consécutives.
- **_context** : Liste des deux dernières interactions.

Méthode `action`

1. **Affichage** : Affiche l'action, la prédiction, le résultat, et la valence de l'interaction précédente.
2. **Enregistrement des deux dernières interactions** : Met à jour `_memory` avec des séquences d'interactions consécutives.
3. **Mise à jour du compteur** : Incrémente `_nb_corrects_in_a_row` en cas de succès consécutif, sinon réinitialise.
4. **Choix d'action pour maximiser la valence** : Choisit la meilleure interaction en fonction de la valence dans `_memory`. Si la valence est négative ou incorrecte, change aléatoirement l'action avec `change_action_outcome`.
5. **Mémorisation de l'interaction** : Enregistre l'interaction prévue pour la prochaine décision.

Autres Méthodes

- **`print_memory`** : Affiche le contenu de `_memory`.
- **`change_action_outcome`** : Change aléatoirement l'action et le résultat.

`Agent4` adapte les actions pour optimiser les résultats en fonction des expériences passées et ajuste ses choix pour maintenir des résultats positifs. Les valences sont positives pour les trois environnements, démontrant l'efficacité de l'agent à apprendre et à s'adapter à des contextes variés. La tortue suit un parcours optimisé pour maximiser les résultats positifs, illustrant la capacité de l'agent à s'adapter à des environnements complexes.

In [120…