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…