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):

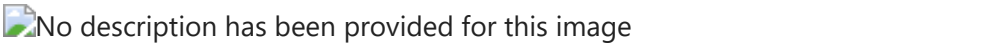
Pour coder notre Agent3 nous avons repris le code pour l'Agent2 et avons introduit une nouvelle action possible (action 2) ainsi que la manière de changer d'action lorsque l'Agent reçoit une valence négative ou s'ennuie. Pour passer d'une action à l'autre nous prennons l'action suivante par exemple si 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.