Université de Picardie Jules Verne
Licence Métiers du Numérique *3ème année*
2021-2022

Paradis et Enfers

EL MEDIOUNI Sohaib

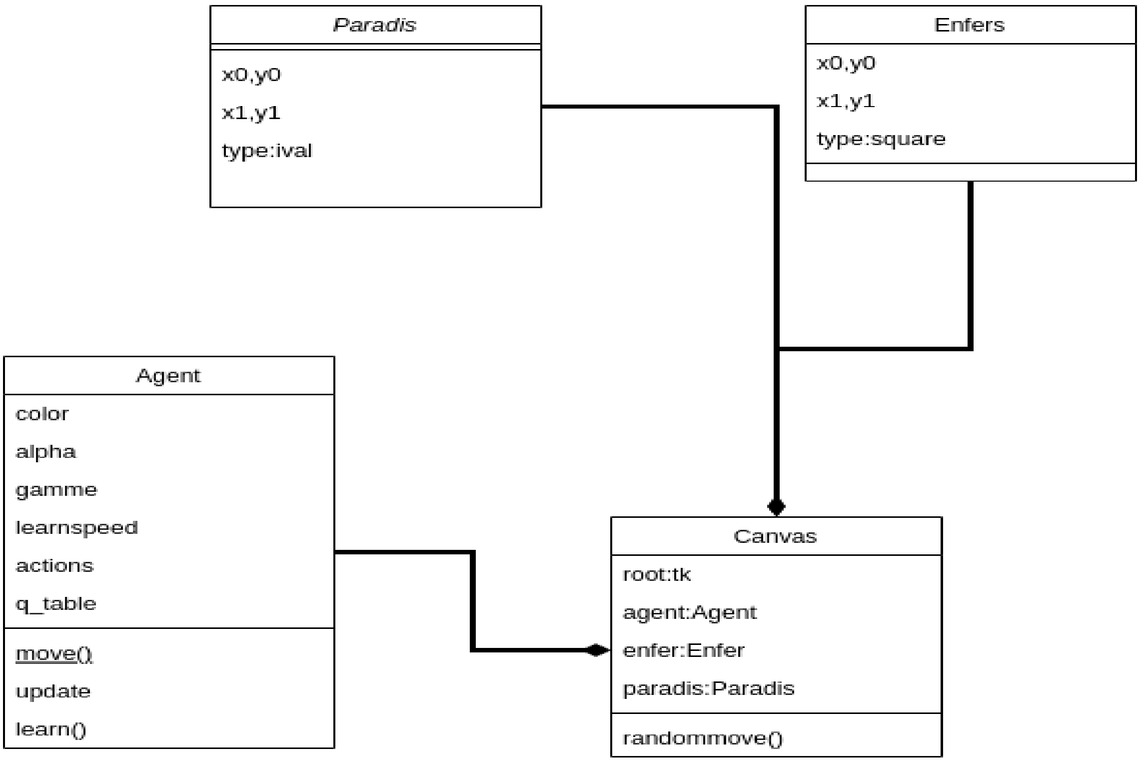
1er avril 2022

Table des matières

Chapitre 1

Introduction

The concept of the game is that the agent must find his path to heaven by avoiding the hell in our case the black squares.



Chapitre 2

Implementation of the code

For the implementation i did a class called canvas that contains an array rows and columns and then i put inside it at the first row the agent and in the 2-3 and 3-2 the hell and the heaven in 3-3 then i make a reintialize() that recreate the agent each time he moves after that I generate a method called randomMoves() where i pick random move based on actions which i have it in an array ('top','bottom','right','left') then i test if it hits the hell i will reward it with -1 and if it hits the heaven i will reward it with 1 and I turn it back (the agent) to the terminal which is the first row and if it is an empty square I make 0 in reward and continue moving. In the class agent I implemented the learning strategy with learn-speed of 0.03 and gamma= influence$_decay_coeff = 0.9 and epsilon = major_prob = 0.9. then I defined the learnstrategy Method that has cur_state, action, reward, next_state and base on the actio$

2.1 Canvas

```
class Canvas(tk.Tk, object):
    def __init__(self):
        super(Canvas, self).__init__()
        self.title("Canvas")
        self.init()

    def init(self):

        self.canvas = tk.Canvas(
        self, bg='white', height=size_height, width=size_width)
        self.actions = ["left", "right", "top", "bottom"]

        #colone
        for c in range(width+1):
            x0, y0 = c * unit, 0
            x1, y1 = x0, y0 + height * unit
            self.canvas.create_line(x0, y0, x1, y1)
        #ligne
        for r in range(width+1):
            x0, y0 = 0, r*unit
            x1, y1 = x0 + width * unit, y0
            self.canvas.create_line(x0, y0, x1, y1)
```

- **Hell :**

```
#First trap HELL 3/2
        x0, y0 = startPoint[0] + (width - 2) * unit, startPoint[1] + (
  height - 3) * unit

        x1, y1 = x0 + 3 * unit / 4, y0 + 3 * unit / 4

        self.hell1 = self.canvas.create_rectangle(x0, y0, x1, y1, fill='
  black')
#second trap HELL 2/3
        x0, y0 = startPoint[0] + (width - 3) * unit, startPoint[1] + (
  height - 2) * unit
        x1, y1 = x0 + 3 * unit / 4, y0 + 3 * unit / 4
        self.hell2 = self.canvas.create_rectangle(x0, y0, x1, y1, fill='
  black')
```

- **Heaven and Agent :**

```
        for i in range(6):
            for j in range(6):
                x0, y0 = startPoint[0]+j*unit, startPoint[1]+i*unit
                x1, y1 = x0+3*unit/4, y0+3*unit/4
                if i == 0 and j == 0:
                    #Agent
                    self.rect = self.canvas.create_rectangle(
                                x0, y0,
                                x1, y1,
                                fill='red')

                elif i == 2 and j == 2:
                    oval_center = startPoint + unit * 2
                    #Heaven 3/3
                    self.oval = self.canvas.create_oval(
                                x0, y0,
                                x1, y1,
                                fill='yellow')
        self.canvas.pack()
```

- **reintialize :**

```
    def reintialize(self):
        self.update()
        time.sleep(0.5)
        self.canvas.delete(self.rect)
        startPoint = np.array([unit/8, unit/8])
        x0, y0 = startPoint[0]+0*unit, startPoint[1]+0*unit
        x1, y1 = x0+3*unit/4, y0+3*unit/4
        self.rect = self.canvas.create_rectangle(
```

```
                x0, y0,
                x1, y1,
                fill='red')
        return self.canvas.coords(self.rect)
```

- **random move :**

```

def random_move(self, action):

        cur_state = self.canvas.coords(self.rect)
        base_action = np.array([0,0])
        if action == 'top': # haut
            if cur_state[1] > unit:
                base_action[1] = base_action[1] - unit
        elif action == 'bottom': # bas
            if cur_state[1] < (height-1)*unit:
                base_action[1] = base_action[1] + unit
        elif action == 'right': # droite
            if cur_state[0] < (width-1)*unit:
                base_action[0] = base_action[0] + unit
        elif action == 'left': # gauche
            if cur_state[0] > unit:
                base_action[0] = base_action[0] - unit

        self.canvas.move(self.rect, base_action[0], base_action[1])
        next_move = self.canvas.coords(self.rect)


        if next_move == self.canvas.coords(self.oval):
            reward = 1
            over = True
            next_move = 'terminal'

            # print("Point gagne!")
        elif next_move == self.canvas.coords(self.hell1) or next_move == self.canvas.coords(self.hell2):
            reward = -1
            over = True
            next_move = 'terminal'
            # print("Point perdu!")
        else:
            reward = 0
            over = False
        return next_move, reward, over
```

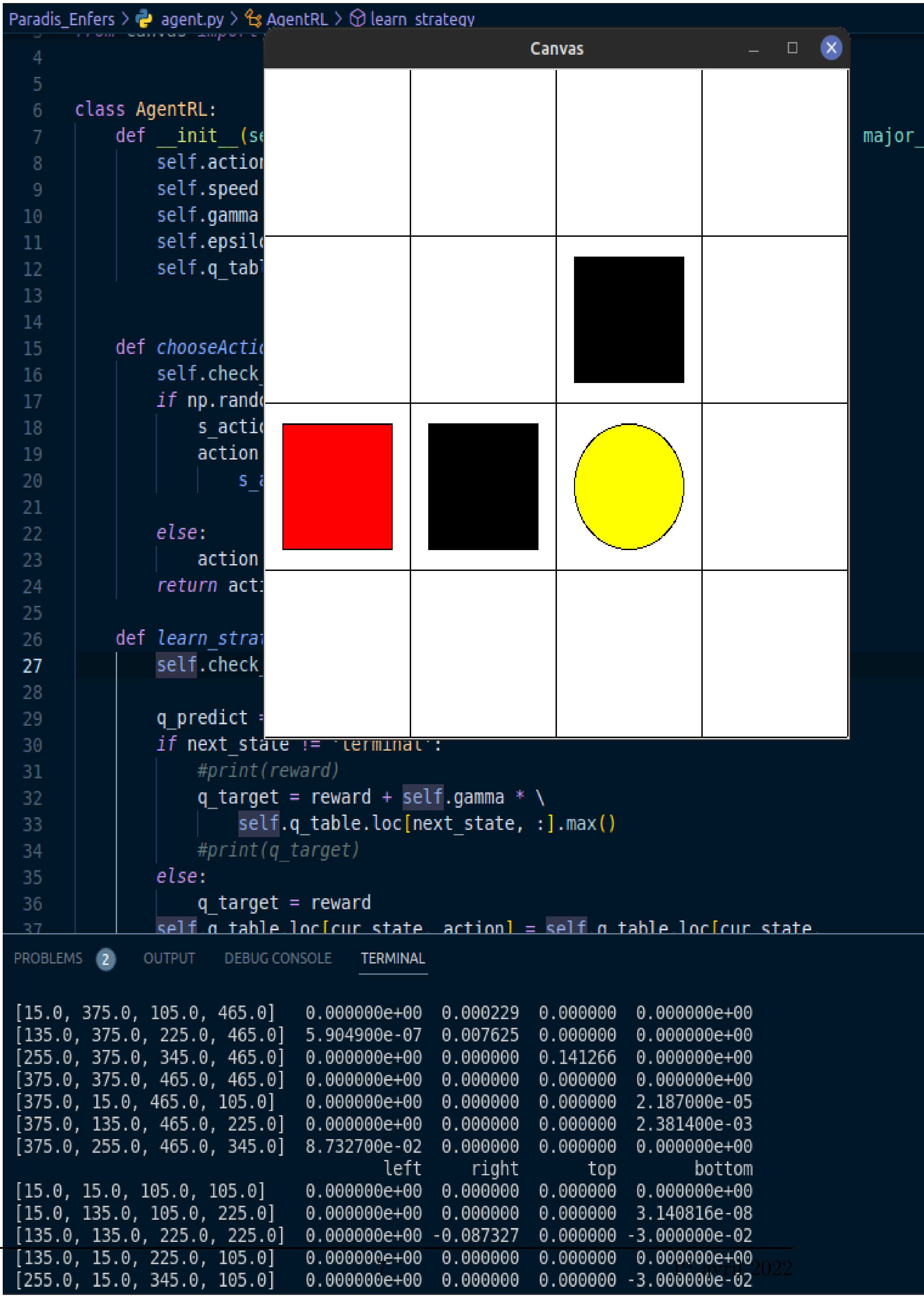
2.2 Agent

- **Learning Strategy :**

```

    def learn_strategy(self, cur_state, action, reward, next_state):
        self.check_state(next_state)

        q_predict = self.q_table.loc[cur_state, action]
        #if the next state is not equal to the first case (terminal)
        if next_state != 'terminal':
            q_target = reward + self.gamma * \
                self.q_table.loc[next_state, :].max()
        else:
            q_target = reward
        self.q_table.loc[cur_state, action] = self.q_table.loc[cur_state
,
                                                                    action]+
self.speed * (q_target - q_predict)
```

- **chooseAction Methode :**

```

    def chooseAction(self, observation):
        self.check_state(observation)
        if np.random.uniform() < self.epsilon:
            s_action = self.q_table.loc[observation, :]
            action = np.random.choice(
                s_action[s_action == np.max(s_action)].index)

        else:
            action = np.random.choice(self.actions)
        return action)
```


Paradis_Enfers > agent.py > AgentRL > learn_strategy
Canvas
class AgentRL:
def __init__(se
major_
self.actior
self.speed
self.gamma
self.epsil
self.q_tabl
def chooseActic
self.check_
if np.rando
s_actic
action
else:
action
return act
def learn_strat
self.check_
q_predict =
if next_state != 'terminal':
#print(reward)
q_target = reward + self.gamma * \
self.q_table.loc[next_state, :].max()
#print(q_target)
else:
q_target = reward
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
[15.0, 375.0, 105.0, 465.0] 0.000000e+00 0.000229 0.000000 0.000000e+00
[135.0, 375.0, 225.0, 465.0] 5.904900e-07 0.007625 0.000000 0.000000e+00
[255.0, 375.0, 345.0, 465.0] 0.000000e+00 0.000000 0.141266 0.000000e+00
[375.0, 375.0, 465.0, 465.0] 0.000000e+00 0.000000 0.000000 0.000000e+00
[375.0, 15.0, 465.0, 105.0] 0.000000e+00 0.000000 0.000000 2.187000e-05
[375.0, 135.0, 465.0, 225.0] 0.000000e+00 0.000000 0.000000 2.381400e-03
[375.0, 255.0, 465.0, 345.0] 8.732700e-02 0.000000 0.000000 0.000000e+00
left right top bottom
[15.0, 15.0, 105.0, 105.0] 0.000000e+00 0.000000 0.000000 0.000000e+00
[15.0, 135.0, 105.0, 225.0] 0.000000e+00 0.000000 0.000000 3.140816e-08
[135.0, 135.0, 225.0, 225.0] 0.000000e+00 -0.087327 0.000000 -3.000000e-02
[135.0, 15.0, 225.0, 105.0] 0.000000e+00 0.000000 0.000000 0.000000e+00
[255.0, 15.0, 345.0, 105.0] 0.000000e+00 0.000000 0.000000 -3.000000e-02