МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЁТ

Рубежный Контроль №__2__
по дисциплине «Методы машинного обучения»

Тема: « »

ИСПОЛНИТЕЛЬ: Доу Линхань__
ФИО
группа ИУ5-21М _____
подпись
"__"_____2023 г.

ПРЕПОДАВАТЕЛЬ: _____
ФИО
_____
подпись
"__"_____2023 г.

Москва - 2023

```python
import numpy as np
import time
import sys
if sys.version_info.major == 2:
    import Tkinter as tk
else:
    import tkinter as tk


UNIT = 40   # pixels
MAZE_H = 4  # grid height
MAZE_W = 4  # grid width


4 用法
class Maze(tk.Tk, object):
    def __init__(self):
        super(Maze, self).__init__()
        self.action_space = ['u', 'd', 'l', 'r']
        self.n_actions = len(self.action_space)
        self.title('maze')
        self.geometry('{0}x{1}'.format(MAZE_W * UNIT, MAZE_H * UNIT))
        self._build_maze()

    1 个用法
    def _build_maze(self):
        self.canvas = tk.Canvas(self, bg='white',
                                height=MAZE_H * UNIT,
                                width=MAZE_W * UNIT)

        # create grids
        for c in range(0, MAZE_W * UNIT, UNIT):
            x0, y0, x1, y1 = c, 0, c, MAZE_H * UNIT
            self.canvas.create_line(x0, y0, x1, y1)
        for r in range(0, MAZE_H * UNIT, UNIT):
            x0, y0, x1, y1 = 0, r, MAZE_W * UNIT, r
            self.canvas.create_line(x0, y0, x1, y1)
```

```
        # create origin
        origin = np.array([20, 20])

        # hell
        hell1_center = origin + np.array([UNIT * 2, UNIT])
        self.hell1 = self.canvas.create_rectangle(
            hell1_center[0] - 15, hell1_center[1] - 15,
            hell1_center[0] + 15, hell1_center[1] + 15,
            fill='black')
        # hell
        hell2_center = origin + np.array([UNIT, UNIT * 2])
        self.hell2 = self.canvas.create_rectangle(
            hell2_center[0] - 15, hell2_center[1] - 15,
            hell2_center[0] + 15, hell2_center[1] + 15,
            fill='black')

        # create oval
        oval_center = origin + UNIT * 2
        self.oval = self.canvas.create_oval(
            oval_center[0] - 15, oval_center[1] - 15,
            oval_center[0] + 15, oval_center[1] + 15,
            fill='yellow')

        # create red rect
        self.rect = self.canvas.create_rectangle(
            origin[0] - 15, origin[1] - 15,
            origin[0] + 15, origin[1] + 15,
            fill='red')

        # pack all
        self.canvas.pack()
```

```
def reset(self):
    self.update()
    time.sleep(0.5)
    self.canvas.delete(self.rect)
    origin = np.array([20, 20])
    self.rect = self.canvas.create_rectangle(
        origin[0] - 15, origin[1] - 15,
        origin[0] + 15, origin[1] + 15,
        fill='red')
    # return observation
    return self.canvas.coords(self.rect)

def step(self, action):
    s = self.canvas.coords(self.rect)
    base_action = np.array([0, 0])
    if action == 0:   # up
        if s[1] > UNIT:
            base_action[1] -= UNIT
    elif action == 1:   # down
        if s[1] < (MAZE_H - 1) * UNIT:
            base_action[1] += UNIT
    elif action == 2:   # right
        if s[0] < (MAZE_W - 1) * UNIT:
            base_action[0] += UNIT
    elif action == 3:   # left
        if s[0] > UNIT:
            base_action[0] -= UNIT

    self.canvas.move(self.rect, base_action[0], base_action[1])  # move agent

    s_ = self.canvas.coords(self.rect)  # next state

    # reward function
    if s_ == self.canvas.coords(self.oval):
        reward = 1
        done = True
        s_ = 'terminal'
```

```
        if s_ == self.canvas.coords(self.oval):
            reward = 1
            done = True
            s_ = 'terminal'
        elif s_ in [self.canvas.coords(self.hell1), self.canvas.coords(self.hell2)]:
            reward = -1
            done = True
            s_ = 'terminal'
        else:
            reward = 0
            done = False

        return s_, reward, done

    def render(self):
        time.sleep(0.1)
        self.update()


def update():
    for t in range(10):
        s = env.reset()
        while True:
            env.render()
            a = 1
            s, r, done = env.step(a)
            if done:
                break

if __name__ == '__main__':
    env = Maze()
    env.after(100, update)
    env.mainloop()
```