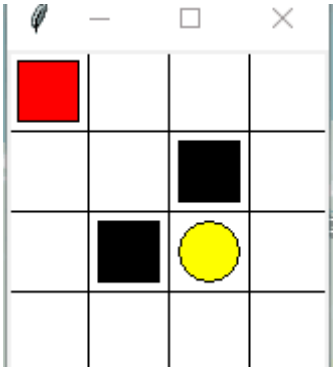
强化学习基础篇（十一）迷宫环境搭建

这节中我们看看如何自己搭建一个强化学习实验环境，这里要做的是一个简单的迷宫环境。智能体在训练过程中的可视化过程如下：



1、环境设定

- 红色正方形表示在探索中的智能体
- 黑色正方形表示一个地狱终结点，当红色智能体到这个状态时，获得-1的奖励。
- 黄色位置表示天堂终结点，当红色智能体到这个状态时，获得+1的奖励。
- 所有其他的白色位置的奖励都为0

2. 源码信息如下

这里我们主要使用numpy, sys以及Tkinter。

```python
# 导入库信息
import numpy as np
import time
import sys
if sys.version_info.major == 2:
    import Tkinter as tk
else:
    import tkinter as tk

# 设定环境信息
UNIT = 40   # 设定是像素大小为40
MAZE_H = 4  # 设置纵轴的格子数量
MAZE_W = 4  # 设置横轴的格子数量

# 创建一个迷宫类
class Maze(tk.Tk, object):
    def __init__(self):
        super(Maze, self).__init__()
        # 定义动作空间为上下左右四个动作
        self.action_space = ['u', 'd', 'l', 'r']
        # 获取动作数量
        self.n_actions = len(self.action_space)
        # 定义迷宫名字
        self.title('maze')
        # 通过geometry函数来设置窗口的宽和高，分别为格子数量乘以像素大小
        self.geometry('{0}x{1}'.format(MAZE_H * UNIT, MAZE_H * UNIT))
        # 调用迷宫创建函数
```

```
        self._build_maze()

    def _build_maze(self):
        # 设定画布大小
        self.canvas = tk.Canvas(self, bg='white',
                                height=MAZE_H * UNIT,
                                width=MAZE_W * UNIT)

        # 创建一个个的小格子
        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)

        # 创建一个原点
        origin = np.array([20, 20])

        # 创建第一个地狱节点
        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')
        # 创建第二个地狱节点
        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')

        # 创建一个圆形的天堂节点
        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')

        # 创建红色探索得智能体
        self.rect = self.canvas.create_rectangle(
            origin[0] - 15, origin[1] - 15,
            origin[0] + 15, origin[1] + 15,
            fill='red')

        # ppack函数的作用是让画布显示中正确的位置上。如果没调用这个函数，就不会正常地显示任何东西。
        self.canvas.pack()
    # 定义重置函数
    def reset(self):
        self.update()
        # 设定重置函数延迟0.5秒
        time.sleep(0.5)
        # 重置的时候删除原来智能体位置，然后重新设定其在原点
        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')
        # 返回重置后的位置
        return self.canvas.coords(self.rect)
    # 定义每步探索函数
    def step(self, action):
        # 首先获取当前的坐标
        s = self.canvas.coords(self.rect)
        # 根据动作来定义智能体会如何移动
        base_action = np.array([0, 0])
        if action == 0:   # 向上移动的情况
            if s[1] > UNIT:
                base_action[1] -= UNIT
        elif action == 1:   # 向下移动的情况
            if s[1] < (MAZE_H - 1) * UNIT:
                base_action[1] += UNIT
        elif action == 2:   # 向又移动的情况
            if s[0] < (MAZE_W - 1) * UNIT:
                base_action[0] += UNIT
        elif action == 3:   # 向左移动的情况
            if s[0] > UNIT:
                base_action[0] -= UNIT
       # 按照动作移动智能体
        self.canvas.move(self.rect, base_action[0], base_action[1])
       # 移动后的状态定义为s_
        s_ = self.canvas.coords(self.rect)

        # 根据移动后的下一个状态，设定奖励值。
        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
        info = None
       # 返回下一个状态，奖励以及当前Episode是否完成的
        return s_, reward, done, info
    # 刷新当前环境
    def render(self):
        time.sleep(0.1)
        self.update()

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

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

3、说明

这个迷宫环境的创建虽然很简单，但是还是按照Gym的框架来搭建的。他包含了一个强化学习环境应有的几个要素，包括以下主要几个函数：

- reset(self):重置环境的状态，返回环境的当前状态。
- step(self, action)：推进一个时间步，返回信息是state, reward, done, info。其中，done用来表明游戏是否结束的标志位，Info是关于游戏的附加信息。
- render（self）：重新绘制环境

4、环境的调用

这里算法的核心后续再介绍，主要看看在运行中是如何对环境进行调用的。

```
def train():
    for episode in range(100):
        # 初始化观测
        observation = env.reset()

        while True:
            # 刷新环境
            env.render()

            # 先通过强化学习算法按照既定策略选择一个动作
            action = RL.choose_action(str(observation))

            # 根据选择的动作，调用环境返回下一个状态，奖励以及结束标志位
            observation_, reward, done = env.step(action)

            # 调用强化学习算法的训练过程
            RL.learn(str(observation), action, reward, str(observation_))

            # 将当前获取的下一步状态设定为新的状态，并准备下一个循环
            observation = observation_

            # 按照标志位结束while
            if done:
                break
    # 游戏结束
    print('game over')
    env.destroy()

if __name__ == "__main__":
    env = Maze()
    RL = QLearningTable(actions=list(range(env.n_actions)))

    env.after(100, train)
    env.mainloop()
```

环境相关的信息就介绍到这里，后续该开始算法部分了。

历史文章链接：

- 强化学习基础篇（十）OpenAI Gym环境汇总
- 强化学习基础篇（九）OpenAI Gym基础介绍
- 强化学习基础篇（八）动态规划扩展
- 强化学习基础篇（七）动态规划之价值迭代
- 强化学习基础篇（六）动态规划之策略迭代（2）
- 强化学习基础篇（五）动态规划之策略迭代（1）
- 强化学习基础篇（四）动态规划之迭代策略评估
- 强化学习基础篇（三）动态规划之基础介绍
- 强化学习基础篇（二）马尔科夫决策过程（MDP）
- 强化学习基础篇（一）强化学习入门
- 9.进一步讨论Policy Gradients方法
- 8. DRL中的Q-Function
- 7. 值函数方法（Value Function Methods）
- 6. Actor-Critic算法
- 5. 策略梯度（Policy Gradients）
- 4.强化学习简介
- 3.TensorFlow示例
- 2.模仿学习（Imitation Learning）
- 1.深度强化学习简介
- A survey on value-based deep reinforcement learning
- Chinese Stock Prediction Using Deep Neural Network
- Differential Dynamics of the Maternal Immune System阅读笔记
- bib如何生成author-year格式的bbl的问题
- 如何使用GPU运行TensorFlow（Win10）
- 通过frp实现内网穿透
- 关于t-SNE降维方法
- 系统评价与Meta分析基础
- Meta分析入门工具介绍
- 使用ruptures检测变量关系
- 如何读取rda格式数据