强化学习：第五次作业

程序源代码文件为 `lovebird.py`，程序的输出（环境界面及两种方法——`exploring_starts`，`on_policy`）放在 `output` 文件夹中。`.avi` 文件是两种方法的寻路展示。由于构建训练数据集以及计算 `Return(s, a)` 的时间较长，因此，我把 `exploring_starts`，`on_policy` 两种方法使用的依据：`q` 值和 `pi` 值，分别放在 `output` 文件夹中的 `q_values.npy` 及 `pi_values.npy`。两种方法的动作决策分别位于 `output` 文件夹中的 `strategy_0.txt` 及 `strategy_1.txt`。

阅读《RL》第五章

主要知道怎样实现Exploring Starts以及On-policy方法：

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \text{argmax}_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

利用蒙特卡洛方法解决鸳鸯找朋友的问题

蒙特卡洛方法通过`MonteCarlo`类实现，类似于上次作业的马尔科夫决策类`Markov`的实现，初始化是需要提供奖赏空间`reward_grid`、动作空间`actions`、折扣因子`gamma`、方法空间`mode_space`及On-policy的参数ϵ，这些参数通过`GameConfig`类进行配置

探索初始化

生成试验数据集

对于`exploring_starts`方法，试验数据集需要一次生成多组试验数据；对于`on_policy`方法，需要在每次迭代的过程中生成一条试验数据。因此，生成多组数据通过`generate_episodes(self, num)`函数实现：

```
def generate_episodes(self, num):
    episodes = []
    for i in range(num):
        for j, state in enumerate(self.s_states):
            episode = self.generate_episode_recurrent(state)
            print(f'{i} epoch, {j} state, {len(episode)}')
            episodes.append(episode)
            return episodes
```

`num`表示需要生成的试验数据的条数。其中，生成一条试验数据通过`generate_episode_recurrent(state)`函数实现。在这里，我分别采用了两种方式来生成一条试验数据，分别参见`lovebird.py`第`411`行和`433`行。

`generate_episode(self, state, episode)`函数是采用递归的方式来生成一条试验数据。在实际应用中，经常会发生递归次数过多而引发`Segement Fault`的错误，因此需要利用`sys.setrecursionlimit(10**6)`命令设置系统最大递归次数。

为了对试验数据集的生成进行更多的控制，`generate_episode_recurrent(self, state)`函数利用循环的方式来生成一条试验数据。在实际应用中，我发现如果不对数据集的生成施加人为控制，某些试验数据的步长会非常长（如：20万步）。因此，为了节省运算时间，在`generate_episode_recurrent(self, state)`函数内，我添加了生成的试验数据最长为1000步的限制：

```
def generate_episode_recurrent(self, state):
    episode = []
    state_init = state.copy()
    while episode == [] or len(episode) > 1000:
```

计算G值及Return值

`G`值和`Return`值的计算通过`calculate_g_first_visit(self, episodes)`函数实现。我采用的是first-visit方法来计算，在计算的同时我也记录了多组数据中某个状态s首次出现的次数`self.return_values_count`：

```
def calculate_g_first_visit(self, episodes):
    for k, episode in enumerate(episodes):
        # print(f'{k} Processing...')
        g = 0
        for i in range(len(episode)-1, -1, -1):
            value = episode[i]
            g = self.i_rewards[value[0][0]][value[0][1]] + g * self.gamma
            if episode.index(value) == i:
                self.return_values[value[0][0]][value[0][1]][action] += g
                self.return_values_count[value[0][0]][value[0][1]][action] += 1
```

计算q值

`q`值计算通过`exploring_starts(self)`函数实现：

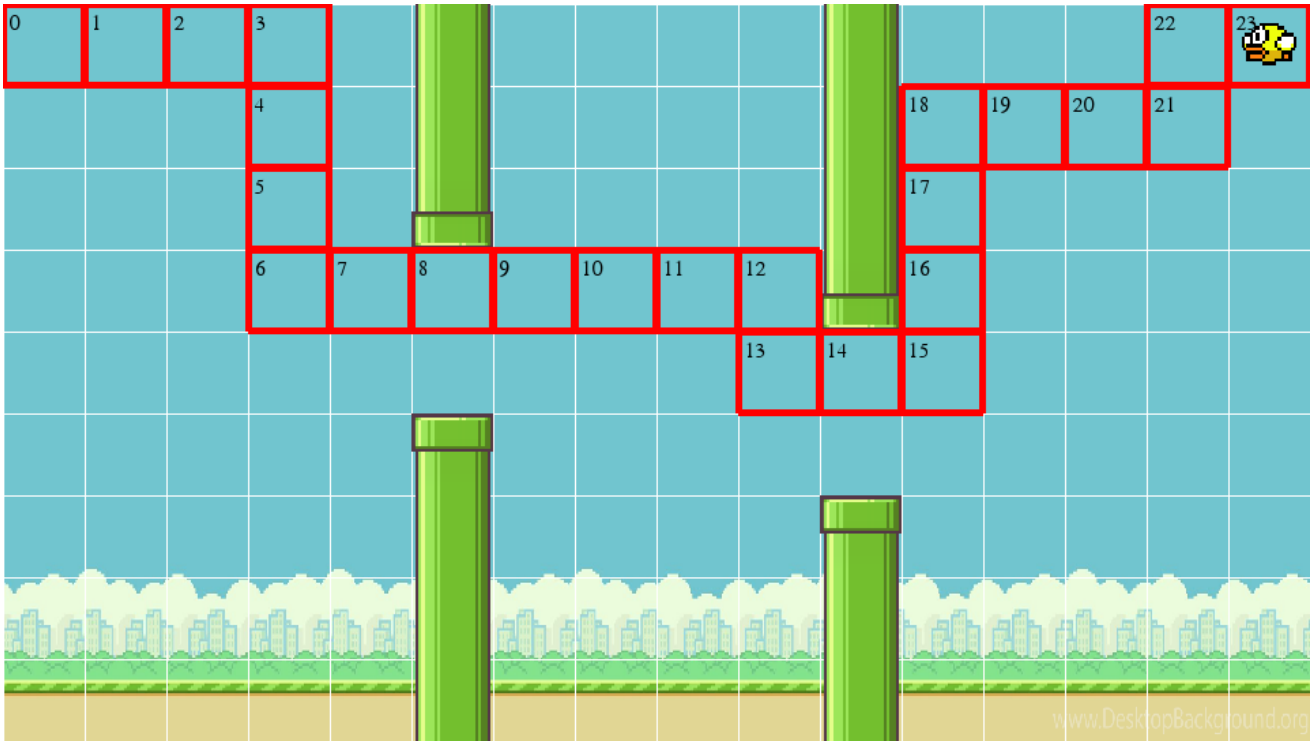
```
q_values[i][j][k] = self.return_values[i][j][k] / self.return_values_count[i][j][k]
```

动作决策

动作决策由`give_action_advice(self, state)`函数实现，对于两种方法，分别使用`q`值和`pi`值来进行动作决策（代码太长，请见源码516行）。

试验结果

`generate_episodes(self, num)`的`num=100`，因此，每个状态都会生成100条试验数据。试验结果如下：



这是我进行许多次试验之后得到的最好的情况。实际操作中，会出现很多种情况：穿柱而过、步数过多、反复横跳、静止不动等等。。。我认为这些情况有的是因为试验数据量不够，有的是因为方法本身造成的。下面的数据表示的是在最好的情况时，进行动作决策时使用的`q`值，以及根据`q`值采用的动作。

```
[-1657.03418384 -1665.69322004 -1669.61790357 -1666.94767545] e
[-1657.82502804 -1665.39490132 -1659.54984846 -1675.09741584] e
[-1648.29459459 -1670.96888101 -1650.70059598 -1656.47513812] e
[-1646.59792645 -1658.66512592 -1644.96462264 -1649.48619247] s
[-1629.44063325 -1648.50525827 -1620.76794833 -1649.60927152] s
[-1610.87841044 -1645.10844283 -1604.82025906 -1640.3187481 ] s
[-1566.722764   -1631.9539823  -1623.03704784 -1631.85246385] e
[-1487.12137134 -1619.08884387 -1584.22704438 -1612.07656613] e
[-1391.63539579 -1573.59768673 -1472.22953216 -1484.40700312] e
[-1333.29010628 -1478.63794708 -1388.26408997 -1366.27153266] e
[-1305.83726628 -1398.66293614 -1328.62508039 -1356.14768944] e
[-1262.33178114 -1339.0084266  -1266.41483272 -1321.44896321] e
[-1257.62747875 -1296.68013872 -1163.12761836 -1298.11372335] s
[-1007.03957399 -1268.82790277 -1162.99433148 -1248.71222644] e
[ -817.91216216 -1174.86587615 -1016.77002875 -1004.25635104] e
[-731.85811633 -996.10040486 -853.85241698 -694.48130004] n
[-650.36007615 -692.09381898 -826.20310055 -587.91162917] n
[-558.68479401 -599.33921772 -687.83685372 -535.7536254 ] n
[-495.18187734 -536.97043853 -599.69633328 -500.5737151 ] e
[-443.71312231 -532.44495251 -572.55001662 -455.7103228 ] e
[-355.44213244 -498.85666372 -522.33983431 -396.37352599] e
[-268.01313198 -442.08645972 -460.52714932 -251.58998669] n
[  -1.         -387.14427229 -367.66197855 -261.89528377] e
```

On Policy

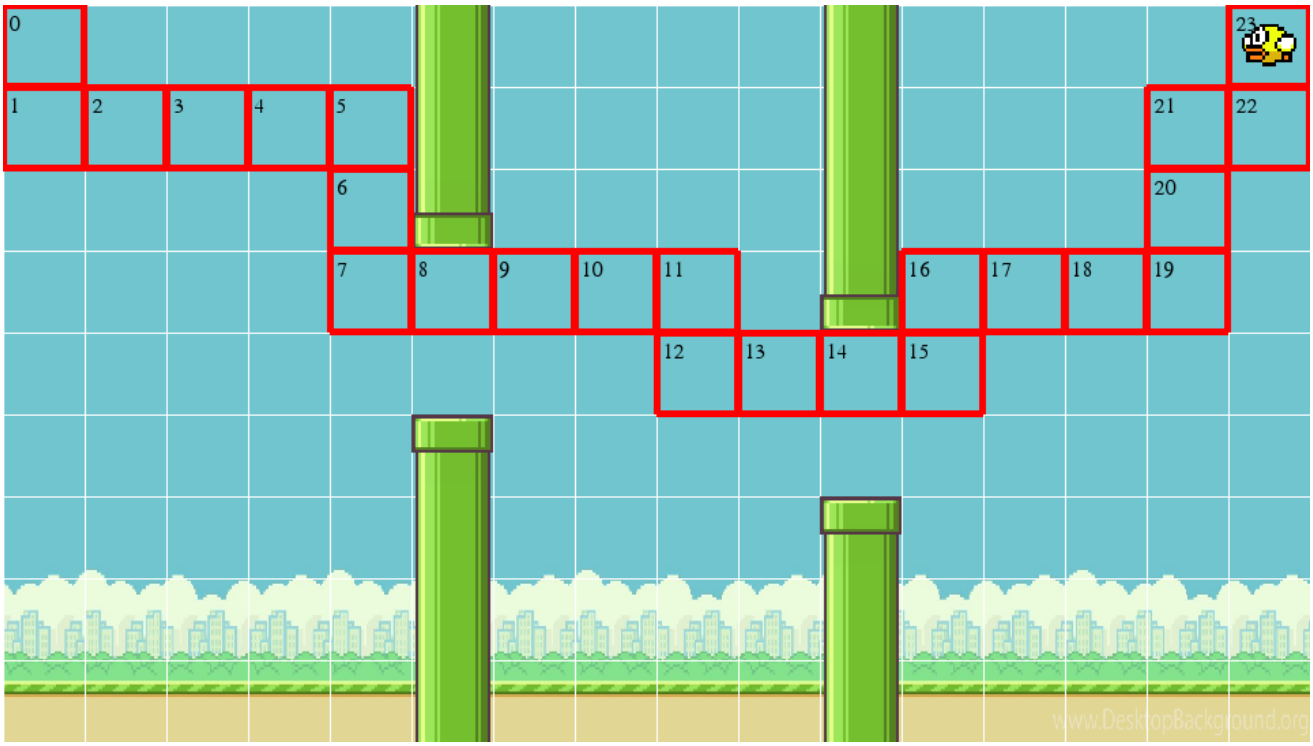
生成试验数据集、计算G值及Return值、动作决策在`探索初始化`方法一节已经提到，在此不再赘述。

计算q值和pi值

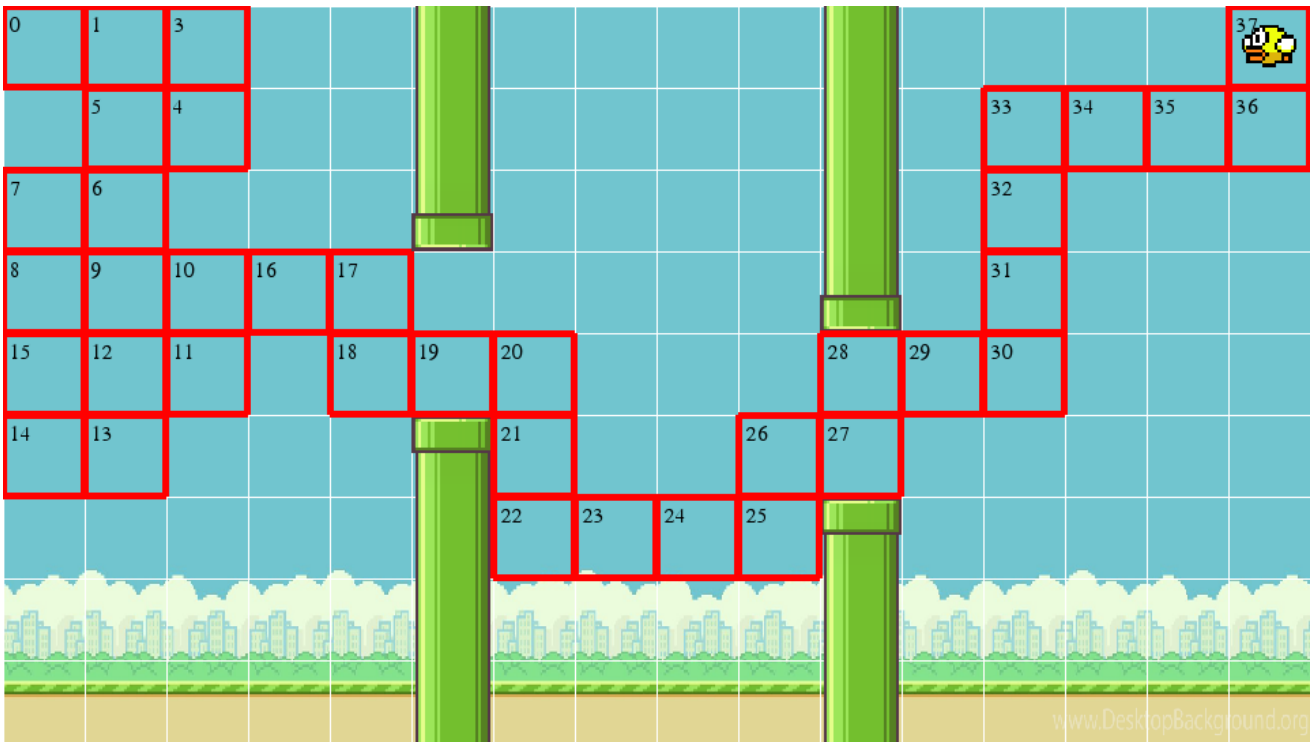
`on_policy(self)`函数用来计算`q`并更新pi值。需要注意，根据书中的定义，每次都需要从S_0, A_0的状态开始按照π来生成试验数据，因此在每次生成数据时，设置`state = [0, 0]`。

试验结果

迭代次数设定为1000：`for c in range(1000)`，ϵ的值为0.1。试验结果如下：



这是我进行许多次试验之后得到的最好的情况。实际操作中，经过学习以后会出现步数过多的情况，比如：



我认为On-Policy方法很大程度上取决于第一次试验数据的生成结果，因为根据第一次的试验数据，π就更新了。后续的动作选择就会依赖于第一次试验数据的结果。

下面的数据表示的是，在最好的情况时，进行动作决策时使用的`pi`值，以及根据`pi`值采用的动作：

```
[0.025 0.025 0.925 0.025] s
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.025 0.025 0.925 0.025] s
[0.025 0.025 0.925 0.025] s
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.025 0.025 0.925 0.025] s
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.025 0.025 0.025 0.925] n
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.925 0.025 0.025 0.025] e
[0.025 0.025 0.025 0.925] n
[0.025 0.025 0.025 0.925] n
[0.925 0.025 0.025 0.025] e
[0.025 0.025 0.025 0.925] n
```

附注

1. 为了方便展示和录制视频，在`change_mode()`函数内，我加载预先处理好的`q`值（`q_values.npy`）和`pi`值（`pi_values.npy`）：

   ```
   if self.mode == 'exploring_starts':
       # self.exploring_starts()
       self.q_values = np.load('./output/q_values.npy').reshape(self.q_values.shape)
       if self.mode == 'on_policy':
       # self.on_policy()
       self.pi_values =
   np.load('./output/pi_values.npy').reshape(self.pi_values.shape)
   ```

2. 此外，希望老师能够对每次作业进行一下作业的说明。比如：

   - 鸟撞到柱子上/边界上需要怎么处理（状态序列、奖赏值）？
   - 算法是否会存在随机性的问题？
   - 学习的结果是否会存在不成功的情况？

   如果不进行这些说明的话，尤其是第一个问题，就会导致在编程的时候存在很大的不确定性，也浪费很多时间。这次作业由于在这些问题上纠缠了太多的时间，我就没有实现off-policy的策略了。