**5b.**

A screenshot of a game

Description automatically generated

**5C**.

**import** numpy **as** np

**import** random

**from** collections **import** defaultdict

**from** environment **import** Env

# Monte Carlo Agent which learns every episodes from the sample

**class** **MCAgent:**

**def** \_\_init\_\_**(**self**,** actions**):**

self**.**width **=** 5

self**.**height **=** 5

self**.**actions **=** actions

self**.**learning\_rate **=** 0.01

self**.**discount\_factor **=** 0.9

self**.**epsilon **=** 0.1

self**.**samples **=** **[]**

self**.**value\_table **=** defaultdict**(float)**

# append sample to memory(state, reward, done)

**def** save\_sample**(**self**,** state**,** reward**,** done**):**

self**.**samples**.**append**([**state**,** reward**,** done**])**

# for every episode, agent updates q function of visited states

**def** update**(**self**):**

G\_t **=** 0

visit\_state **=** **[]**

**for** reward **in** **reversed(**self**.**samples**):**

state **=** **str(**reward**[**0**])**

**if** state **not** **in** visit\_state**:**

visit\_state**.**append**(**state**)**

G\_t **=** self**.**discount\_factor **\*** **(**reward**[**1**]** **+** G\_t**)**

value **=** self**.**value\_table**[**state**]**

self**.**value\_table**[**state**]** **=** **(**value **+**

self**.**learning\_rate **\*** **(**G\_t **-** value**))**

# get action for the state according to the q function table

# agent pick action of epsilon-greedy policy

**def** get\_action**(**self**,** state**):**

**if** np**.**random**.**rand**()** **<** self**.**epsilon**:**

# take random action

valid\_actions **=** self**.**possible\_actions**(**state**)**

action **=** np**.**random**.**choice**(**self**.**actions**)**

**else:**

# take action according to the q function table

next\_state **=** self**.**possible\_next\_state**(**state**)**

action **=** self**.**arg\_max**(**next\_state**)**

**return** **int(**action**)**

**def** possible\_actions**(**self**,** state**):**

col**,** row **=** state

actions **=** **[]**

**if** row **!=** 0**:**

actions**.**append**(**0**)** # Up

**if** row **!=** self**.**height **-** 1**:**

actions**.**append**(**1**)** # Down

**if** col **!=** 0**:**

actions**.**append**(**2**)** # Left

**if** col **!=** self**.**width **-** 1**:**

actions**.**append**(**3**)** # Right

**return** actions

# compute arg\_max if multiple candidates exit, pick one randomly

@staticmethod

**def** arg\_max**(**next\_state**):**

max\_index\_list **=** **[]**

max\_value **=** next\_state**[**0**]**

**for** index**,** value **in** **enumerate(**next\_state**):**

**if** value **>** max\_value**:**

max\_index\_list**.**clear**()**

max\_value **=** value

max\_index\_list**.**append**(**index**)**

**elif** value **==** max\_value**:**

max\_index\_list**.**append**(**index**)**

**return** random**.**choice**(**max\_index\_list**)**

# get the possible next states

**def** possible\_next\_state**(**self**,** state**):**

col**,** row **=** state

next\_state **=** **[**0.0**]** **\*** 4

**if** row **!=** 0**:**

next\_state**[**0**]** **=** self**.**value\_table**[str([**col**,** row **-** 1**])]**

**else:**

next\_state**[**0**]** **=** self**.**value\_table**[str(**state**)]**

**if** row **!=** self**.**height **-** 1**:**

next\_state**[**1**]** **=** self**.**value\_table**[str([**col**,** row **+** 1**])]**

**else:**

next\_state**[**1**]** **=** self**.**value\_table**[str(**state**)]**

**if** col **!=** 0**:**

next\_state**[**2**]** **=** self**.**value\_table**[str([**col **-** 1**,** row**])]**

**else:**

next\_state**[**2**]** **=** self**.**value\_table**[str(**state**)]**

**if** col **!=** self**.**width **-** 1**:**

next\_state**[**3**]** **=** self**.**value\_table**[str([**col **+** 1**,** row**])]**

**else:**

next\_state**[**3**]** **=** self**.**value\_table**[str(**state**)]**

**return** next\_state

# main loop

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

env **=** Env**()**

agent **=** MCAgent**(**actions**=list(range(**env**.**n\_actions**)))**

**for** episode **in** **range(**1000**):**

state **=** env**.**reset**()**

action **=** agent**.**get\_action**(**state**)**

**while** **True:**

env**.**render**()**

# forward to next state. reward is number and done is boolean

next\_state**,** reward**,** done **=** env**.**step**(**action**)**

agent**.**save\_sample**(**next\_state**,** reward**,** done**)**

# get next action

action **=** agent**.**get\_action**(**next\_state**)**

# at the end of each episode, update the q function table

**if** done**:**

**print(**"episode : "**,** episode**)**

**print(**"reward at episode %d: %d" **%** **(**episode**,** reward**))**

agent**.**update**()**

agent**.**samples**.**clear**()**

**break**

# main loop

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

env **=** Env**()**

agent **=** MCAgent**(**actions**=list(range(**env**.**n\_actions**)))**

**for** episode **in** **range(**1000**):**

state **=** env**.**reset**()**

action **=** agent**.**get\_action**(**state**)**

**while** **True:**

env**.**render**()**

# forward to next state. reward is number and done is boolean

next\_state**,** reward**,** done **=** env**.**step**(**action**)**

agent**.**save\_sample**(**next\_state**,** reward**,** done**)**

# get next action

action **=** agent**.**get\_action**(**next\_state**)**

# at the end of each episode, update the q function table

**if** done**:**

**print(**"episode : "**,** episode**)**

**print(**"reward at episode %d: %d" **%** **(**episode**,** reward**))**

agent**.**update**()**

agent**.**samples**.**clear**()**

**break**