**PLAYING WITH REWARD FUNCTION, EXPLORATION\_RATE AND LEARNING\_RATE:**

1. 4 levels of reward with a gap between -0.3 and -0.2

GAMMA = 0.98

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.99995

def get\_reward(state):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

if state[0]<-0.7:

return ((state[0]+0.7))

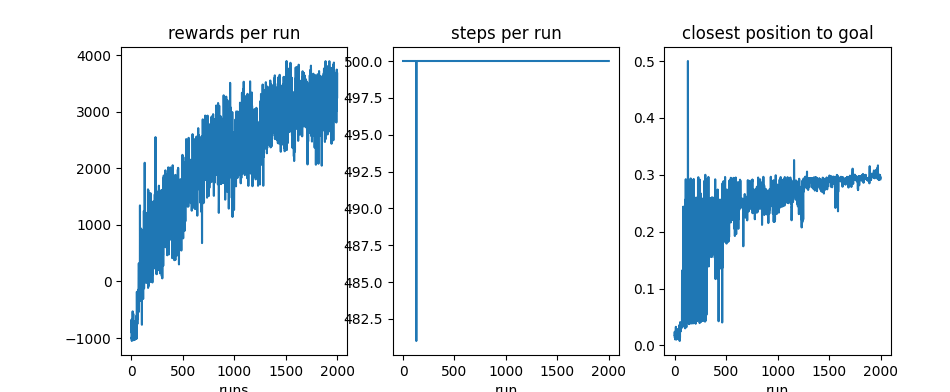
if state[0]>-0.7 and state[0]<-0.3:

return 9\*(state[0]+0.3)

if state[0]>-0.2:

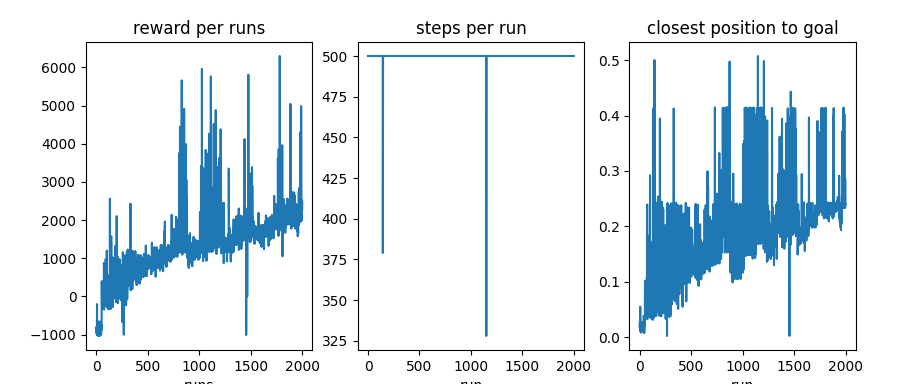
return (9\*(state[0]+0.3))\*\*2

return 0



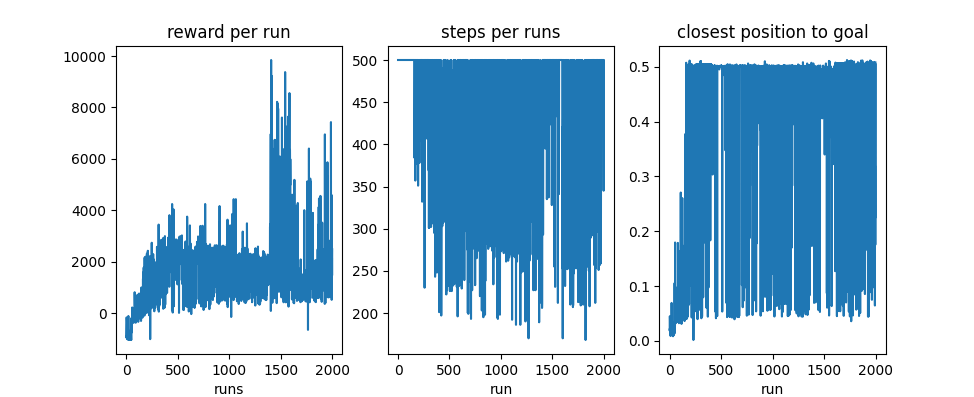
1. Same than previous but less decay rate => more exploration time

EXPLORATION\_MIN = 0.9995

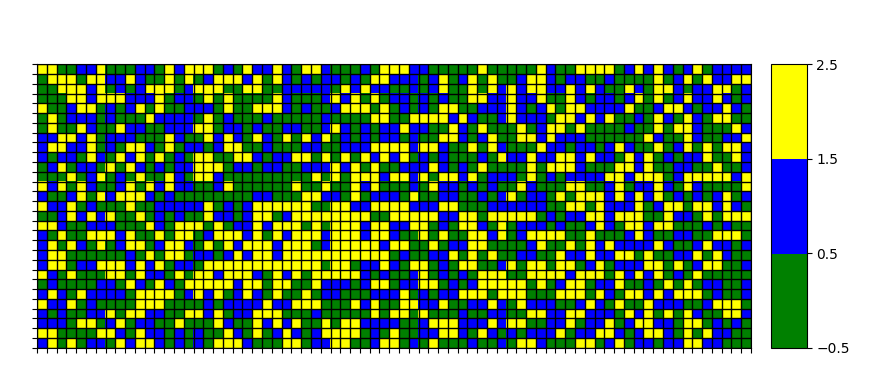


Same than previous one but with minimum exploration of 0.01 THIS IS THE BEST!!!!!

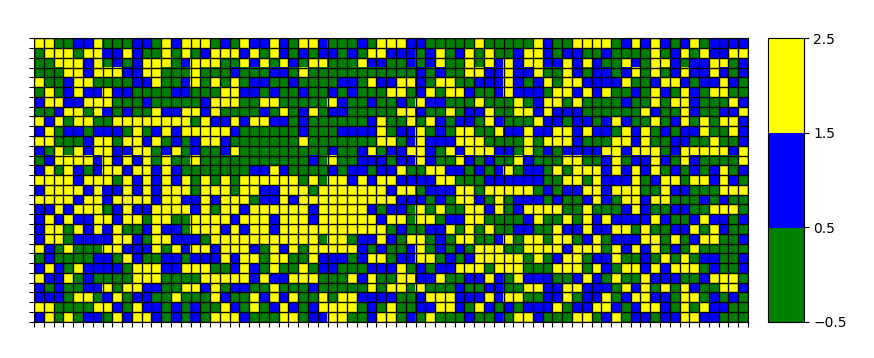
\*Note however that not always are that good.



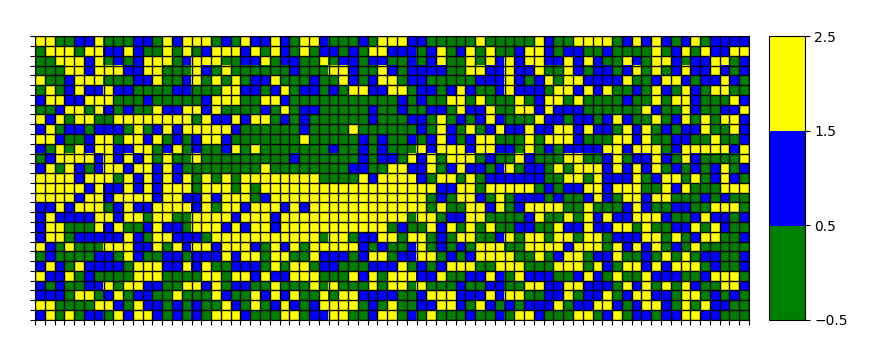
Episode 200



Episode 600



Episode 2000



1. Same than previous one but without the gap and minimum exploration of 0.05

EXPLORATION\_MIN = 0.05

def get\_reward(state, step):

if state[0] >= 0.5:

return 500

if state[0]<-0.7:

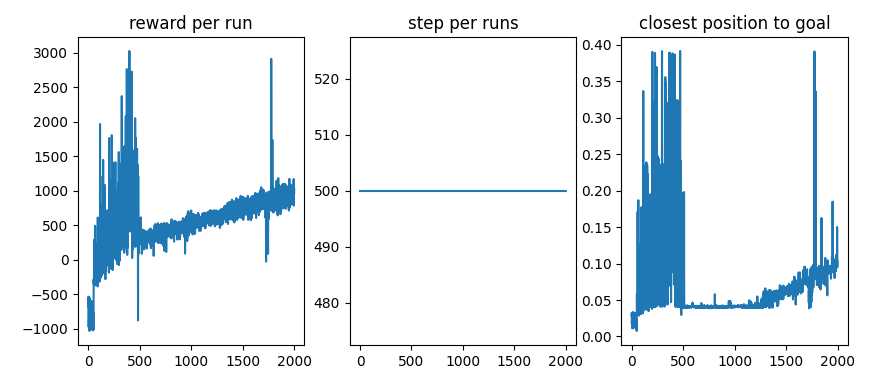
return ((state[0]+0.7))

if state[0]>-0.7 and state[0]<-0.2:

return 9\*(state[0]+0.3)

if state[0]>-0.2:

return (9\*(state[0]+0.3))\*\*2



Trying a softer configuration to converge more securely and a more sophisticated reward function to try to make it faster (avoiding extra steps)

GAMMA = 0.95

LEARNING\_RATE = 0.15

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.9995

def get\_reward(state, step):

if state[0] >= 0.5:

reward= 500

else if state[0]<-0.7:

reward=((state[0]+0.7))

else if state[0]>-0.7 and state[0]<-0.4:

reward= 3\*(state[0])

else if state[0]>-0.4 and state[0]<-0.2:

reward= (9\*(state[0]+0.4))

else if state[0]>-0.2 and state[0]<0.2

reward= (9\*(state[0]+0.5))\*\*2

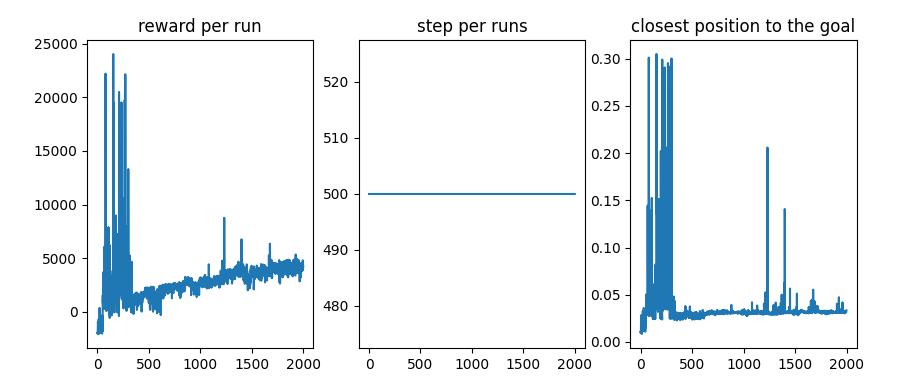
else if state[0]>0.2:

reward= state[0]\*500

if step>100:

reward=reward\*(step/100)

return reward



1. 3 levels of reward with penalty to number of iterations

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

# if state[0] > -0.4:

# return (1+state[0])\*\*2

# if state[0] <-0.49 or state[0]>-0.51:

# return -0.2

# return 0

if state[0]<-0.7:

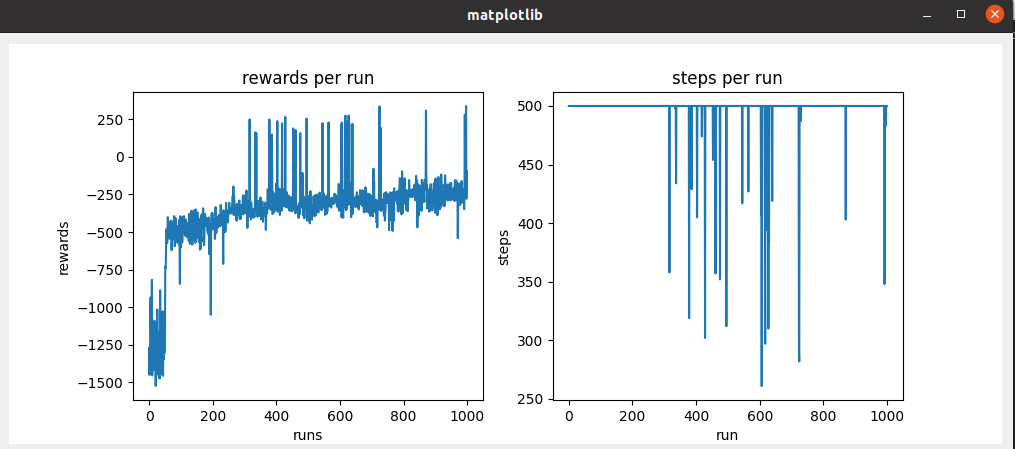
return ((state[0]+0.7))/(step/2)

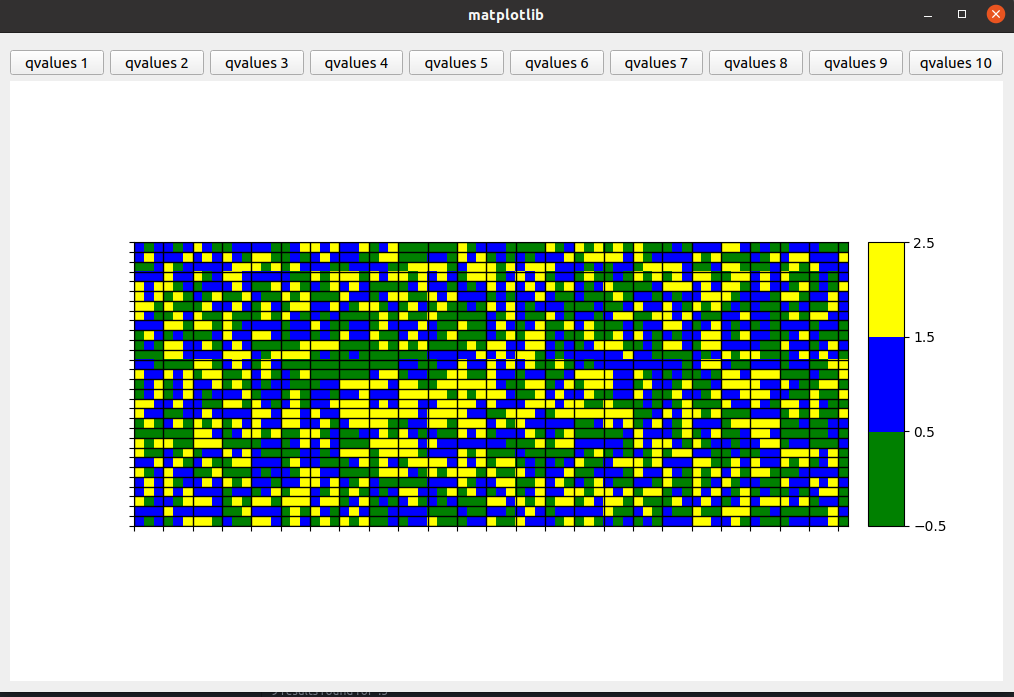
if state[0]>=-0.7 and state[0]<=-0.2:

return 9\*(state[0]+0.2)

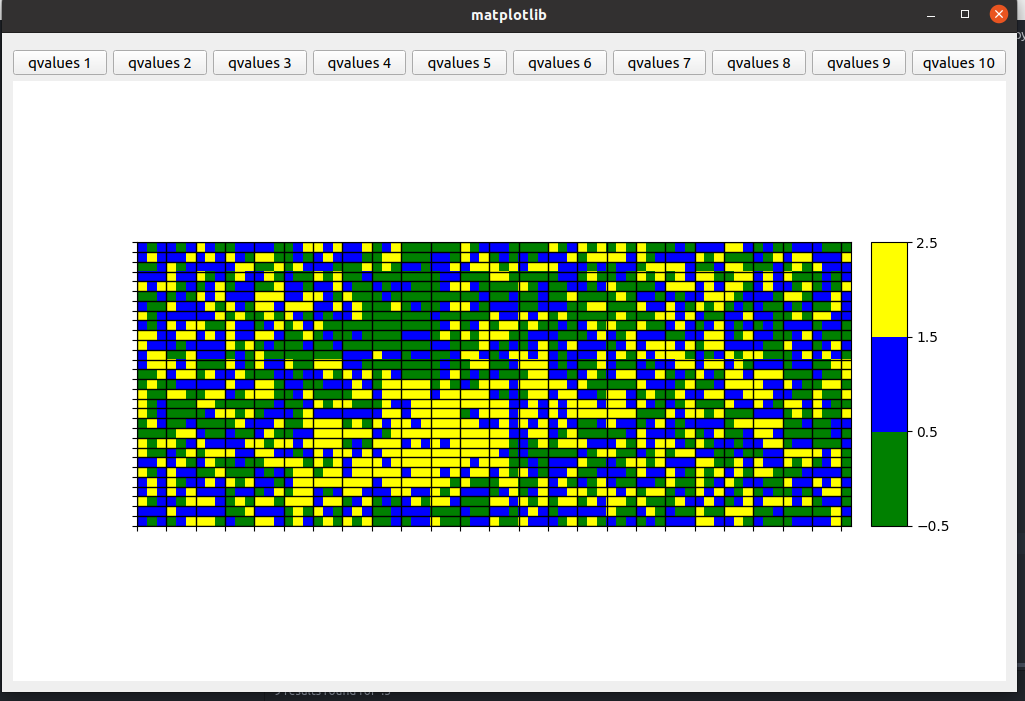
if state[0]>-0.2:

return (9\*(state[0]+0.3))\*\*2/(step/2)

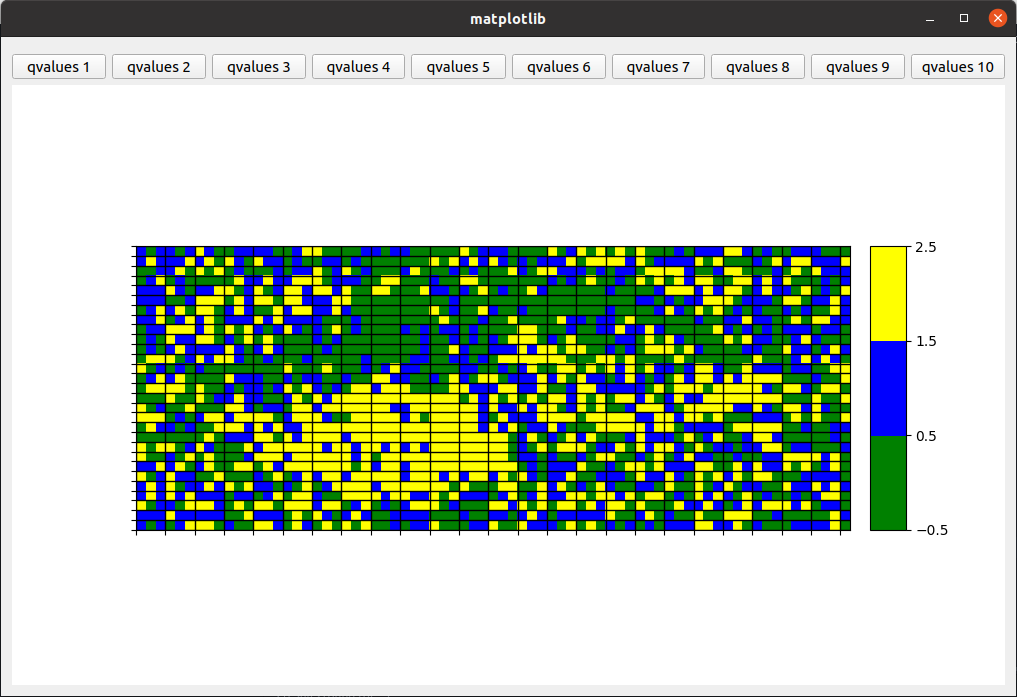


Episode 100

Episode 500



Episode 1000



1. 3 Level of rewards with a higher final reward when steps to achieve that reward are less

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 250+(150\*((MAXIMUM\_STEPS-step)/100))

if state[0]<-0.7:

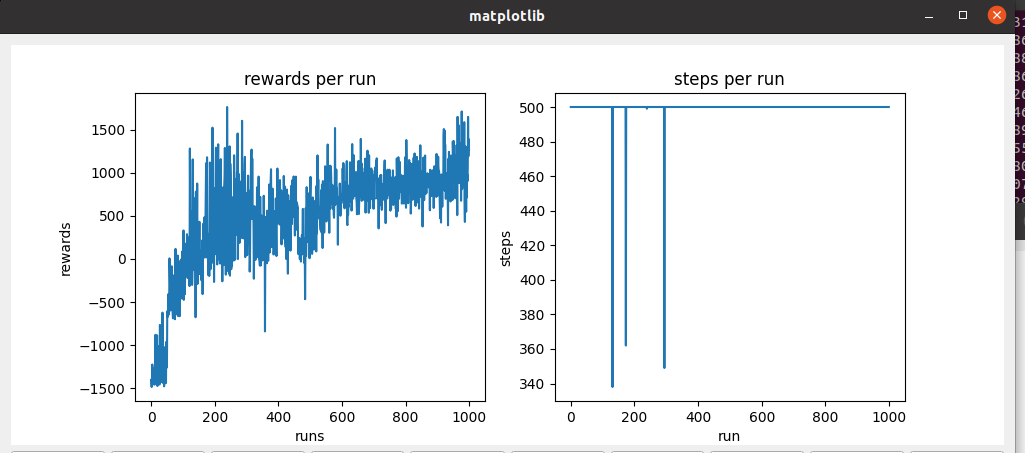
return ((state[0]+0.7))

if state[0]>=-0.7 and state[0]<=-0.2:

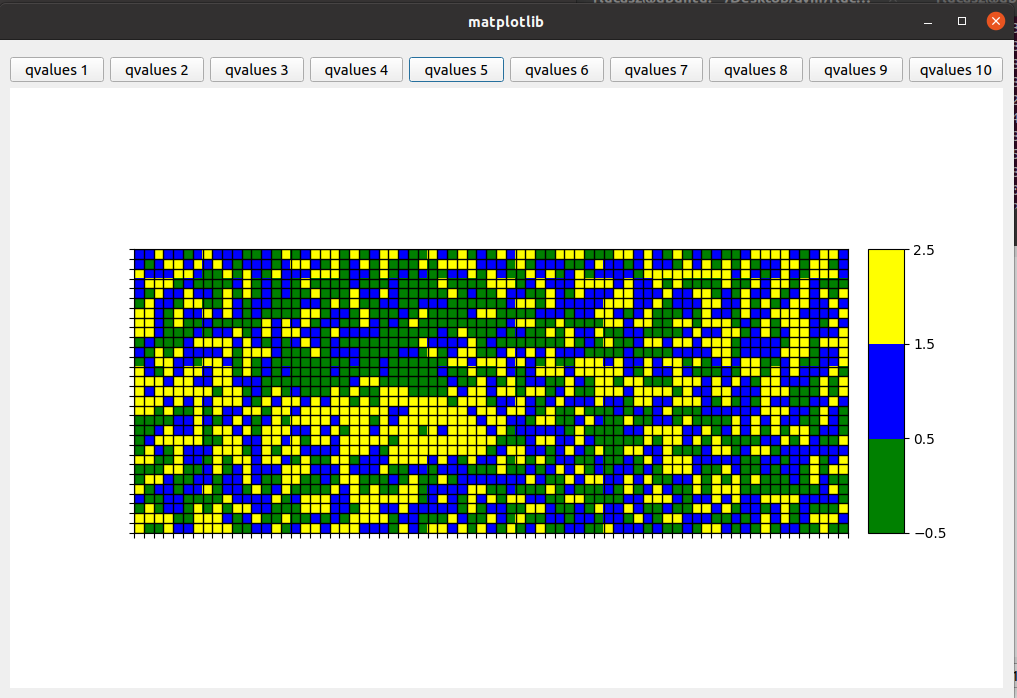
return 9\*(state[0]+0.2)

if state[0]>-0.2:

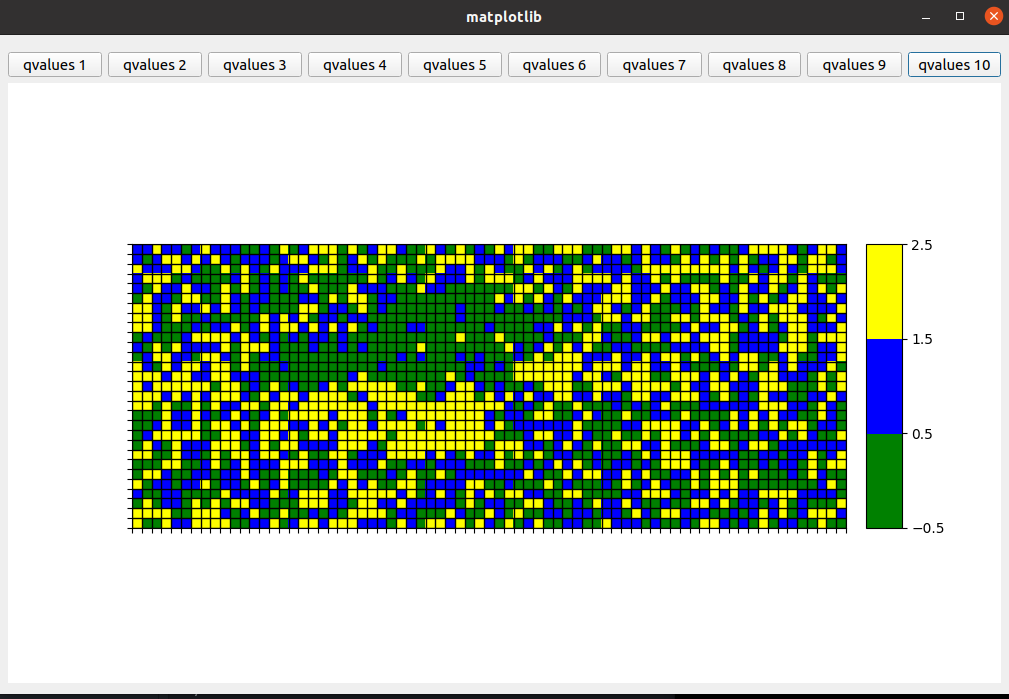
return (9\*(state[0]+0.3))\*\*2



Episode 500



Episode 1000



1. 3 Level of rewards giving higher reward as soon as a higher position is achieved sooner. (in the previous examples) and less decay of exploration\_rate.

GAMMA = 0.98

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.1

EXPLORATION\_DECAY = 0.99995

def get\_reward(state, step):

if state[0] >= 0.5:

return 250+(150\*((MAXIMUM\_STEPS-step)/100))

if state[0]<-0.7:

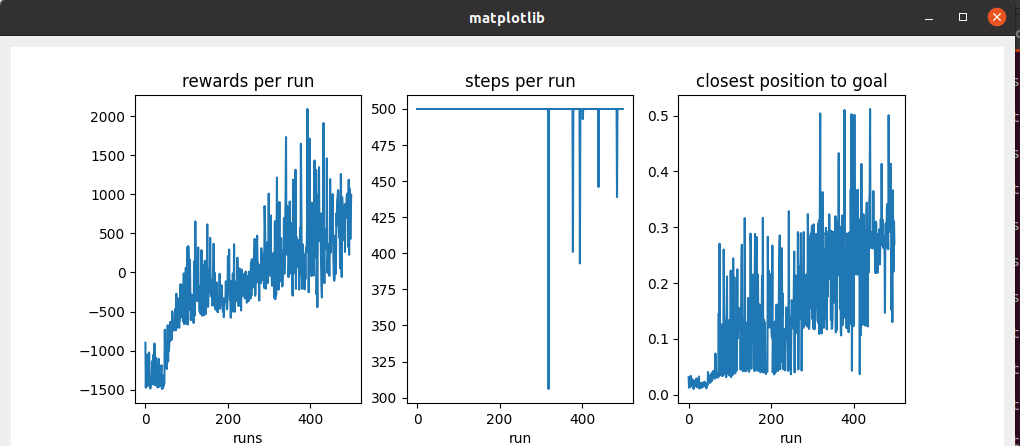
return ((state[0]+0.7))

if state[0]>=-0.7 and state[0]<=-0.1:

return 9\*(state[0]+0.2)

if state[0]>-0.2:

return (4\*((MAXIMUM\_STEPS-step)/100)\*(state[0]+0.3))\*\*2



1. 3 Level of rewards giving higher reward as soon as a higher position is achieved sooner. It is the same than previous but now the learning rate is now 0.1 instead of 0.2, the exploration rate is lower again and the premium level is sooner (-0.3 instead of -0.2).

GAMMA = 0.98

LEARNING\_RATE = 0.1

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.02

EXPLORATION\_DECAY = 0.9995

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 250+(150\*((MAXIMUM\_STEPS-step)/100))

if state[0]<-0.7:

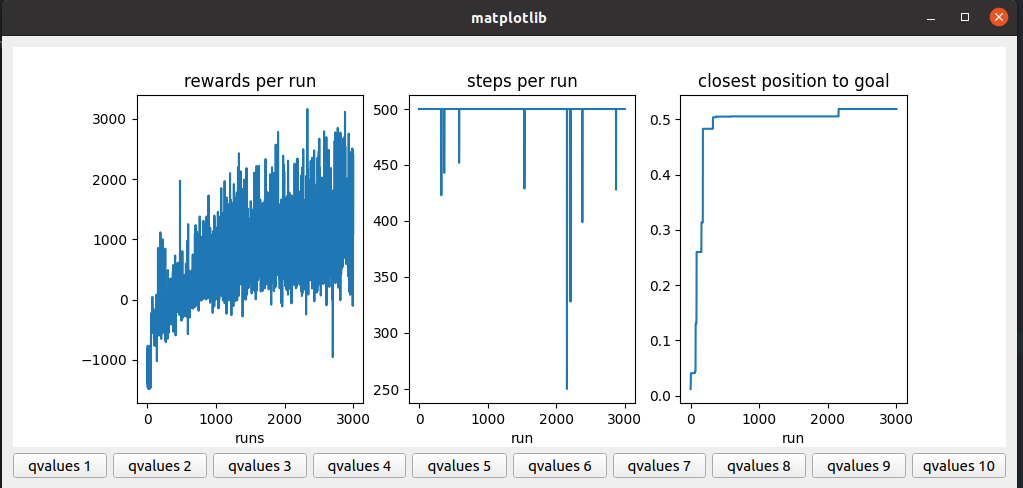
return ((state[0]+0.7))

if state[0]>=-0.7 and state[0]<=-0.3:

return 9\*(state[0]+0.2)

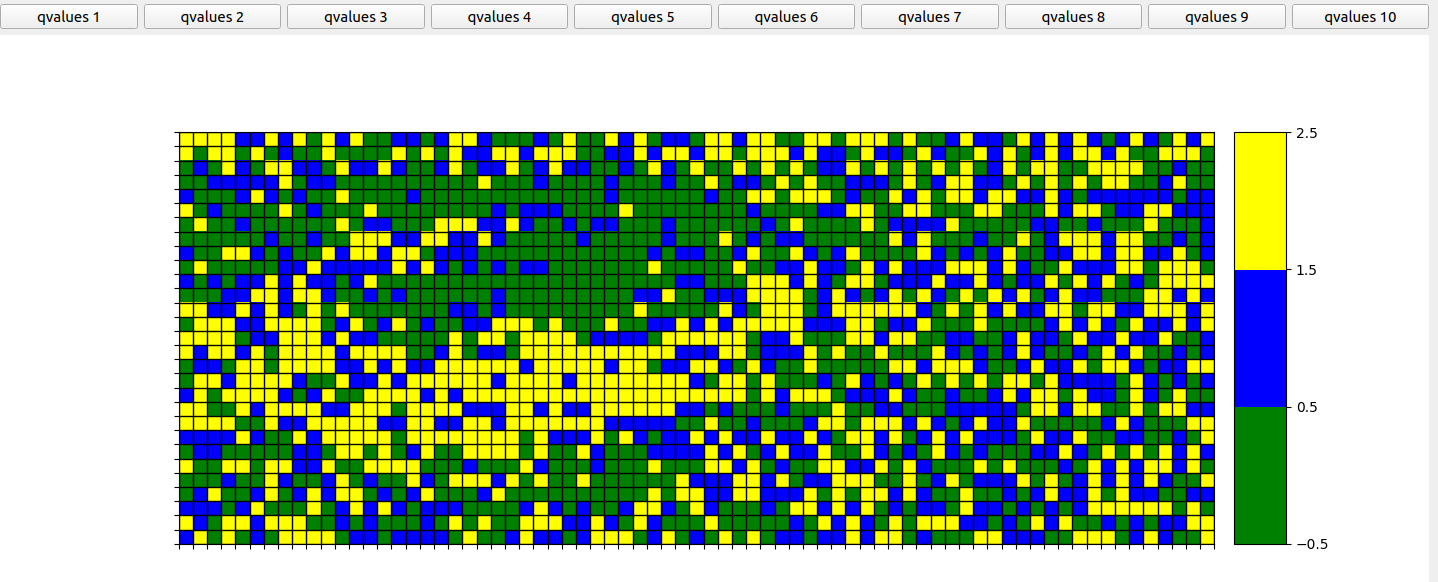
if state[0]>-0.3:

return (4\*((MAXIMUM\_STEPS-step)/100)\*(state[0]+0.3))\*\*2

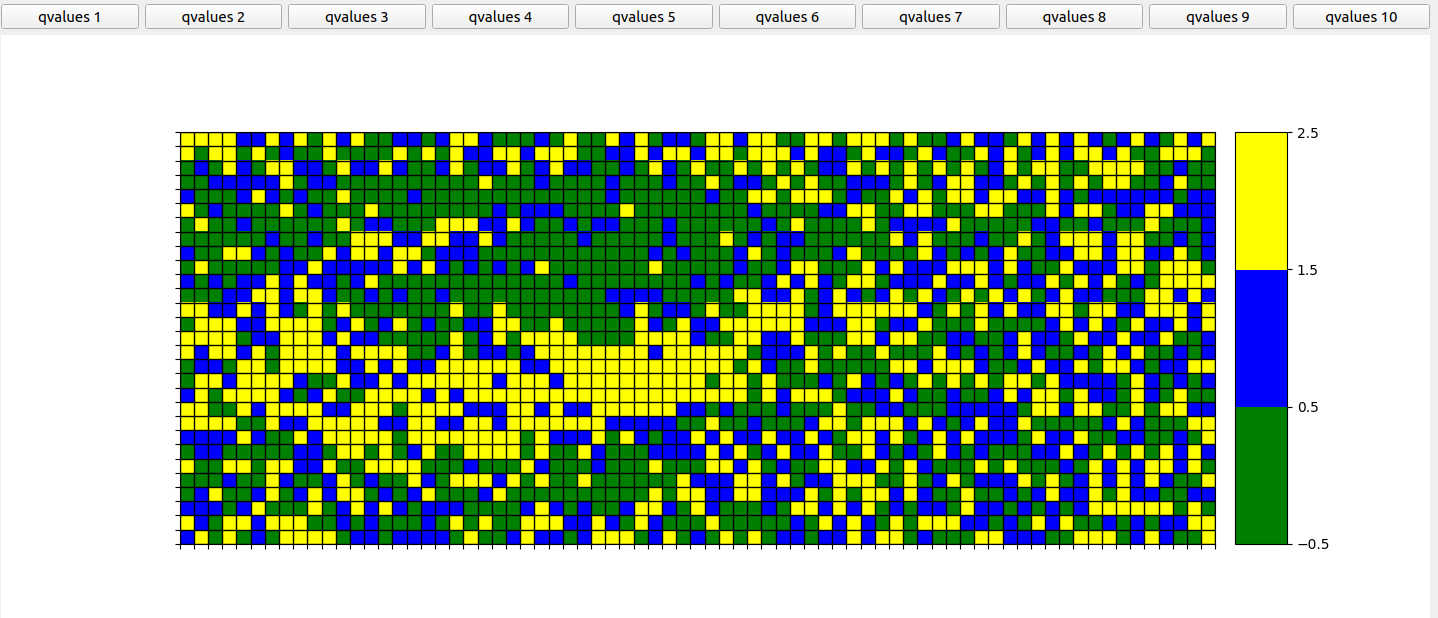


Note that, instead of looking like the goal is reached in last iterations, the goal is not reached when 500 episodes are shown in second graph.

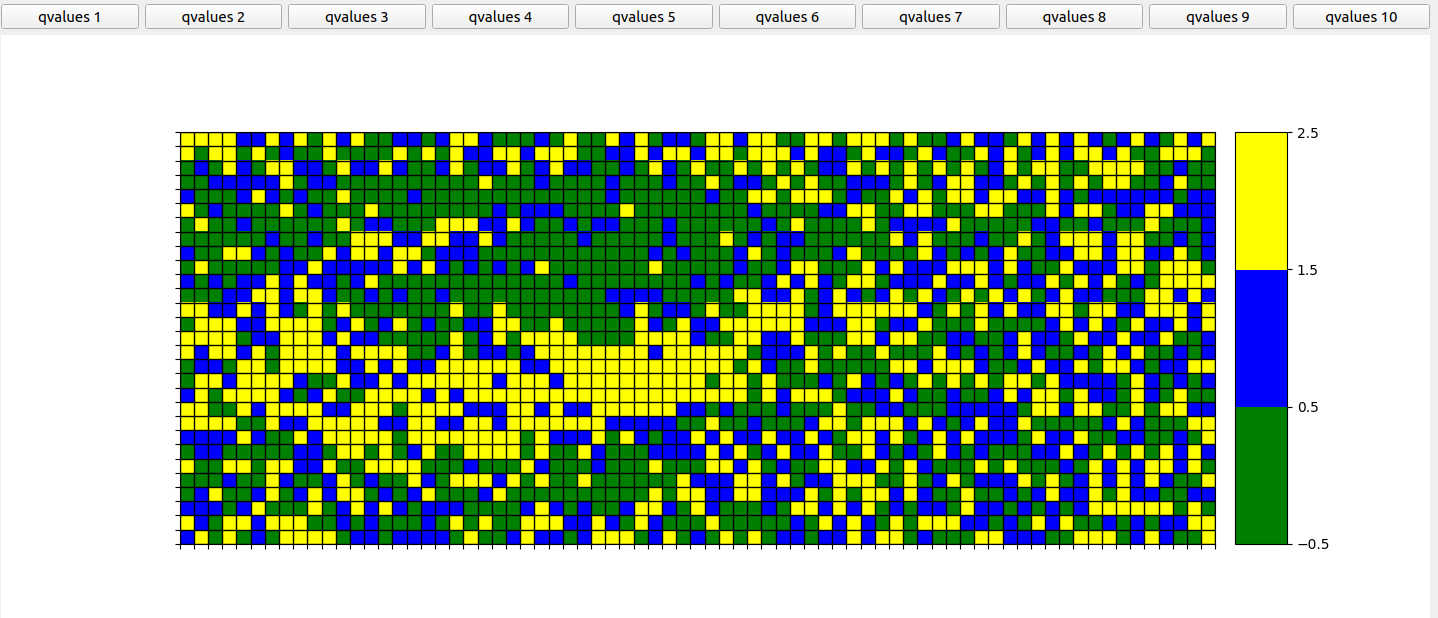
Episode 1500



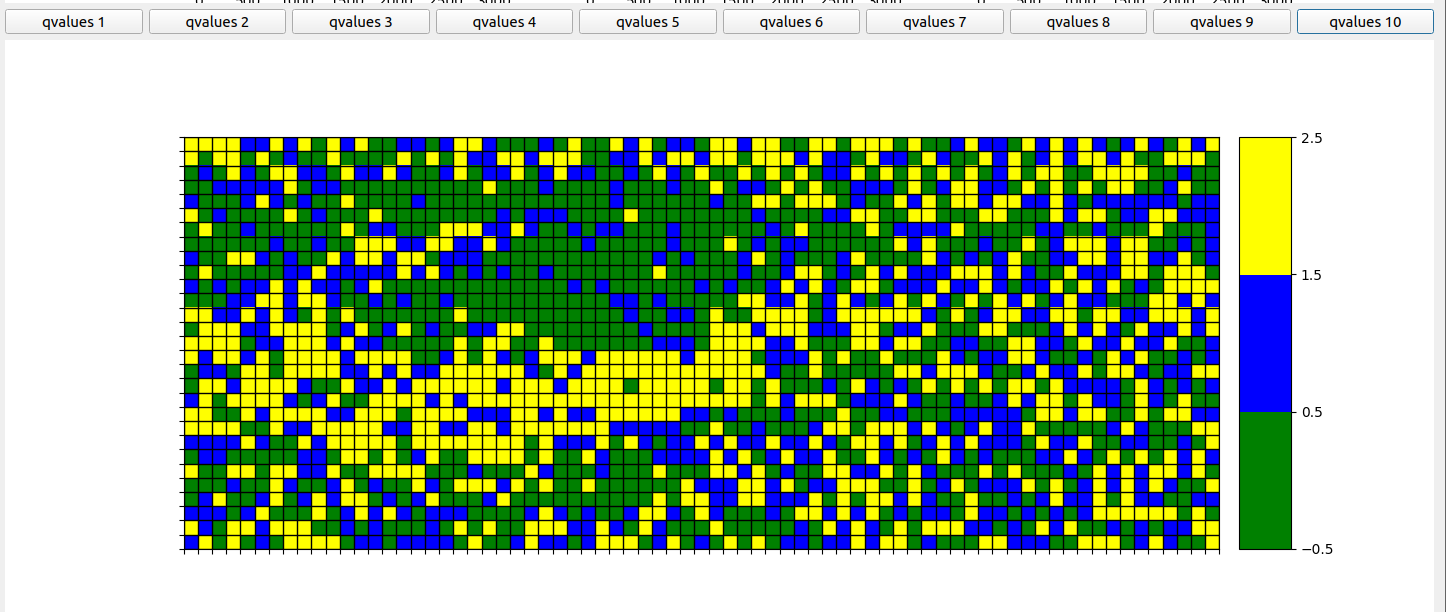
Episode 1800



Episode 2100



Episode 3000



1. 4 levels of reward.

GAMMA = 0.98

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.1

EXPLORATION\_DECAY = 0.99995

def get\_reward(state, step):

if state[0] >= 0.5:

return 500

if state[0]<-0.7:

return ((state[0]+0.7))

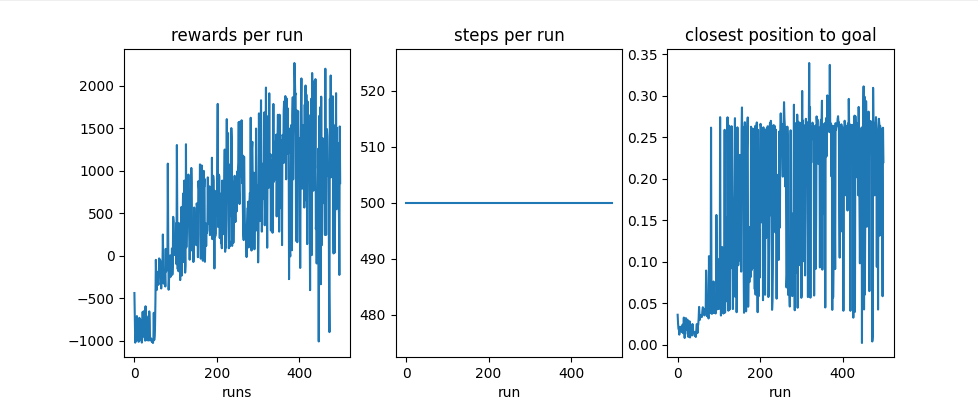
if state[0]>-0.7 and state[0]<-0.3:

return 9\*(state[0]+0.3)

if state[0]>-0.2:

return (9\*(state[0]+0.3))\*\*2

return 0



1. 4 levels of exclusive rewards and different levels of exploration according to the times a state has happened.

GAMMA = 0.98

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.01

EXPLORATION\_DECAY = 0.9995

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

elif state[0]<=-0.7:

return ((state[0]+2))

elif state[0]>-0.7 and state[0]<=-0.3:

return 9\*(state[0]+0.3)

elif state[0]>-0.3 and state[0]<=0:

return (9\*(state[0]+2))

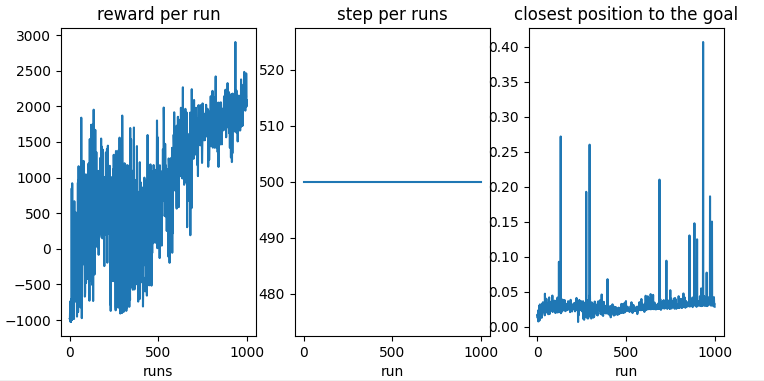
if state[0]>0:

return (9\*state[0]+2)\*\*2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

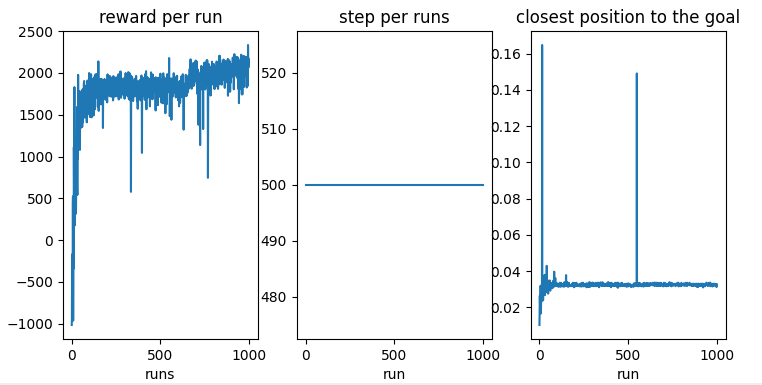
self.exploration\_rate[state\_adj[1]][state\_adj[0]] \*= EXPLORATION\_DECAY

self.exploration\_rate[state\_adj[1]][state\_adj[0]] = max(EXPLORATION\_MIN, self.exploration\_rate[state\_adj[1]][state\_adj[0]])



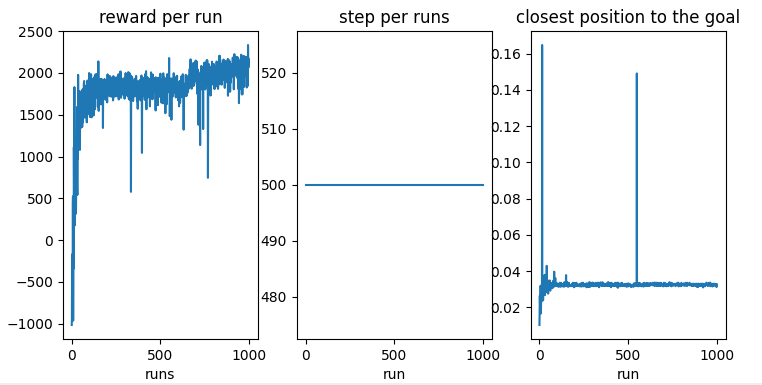
Same than previous one but decreasing the exploration\_decay. In previous case some states remained in a high exploration rate forever.

EXPLORATION\_DECAY = 0.95



Same than previous one but decreasing even more the exploration\_decay. In previous case some states remained in a high exploration rate forever.

EXPLORATION\_DECAY = 0.8



**Some attempts later I realized that some of the previous tried configurations last at least 2000 runs to converge. It is possible that we are lucky with the randomness of the learning and it converge much earlier, but if we want to make sure we need to increase the number of runs. We tried the following configurations with success.**

A higher learning rate and a gap between some rewards levels.

MAX\_RUNS=3000

MAXIMUM\_STEPS=500

INTERPOLATION=MAX\_RUNS/10

GAMMA = 0.95

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.9995

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

if state[0]<-0.7:

return ((state[0]+0.7))

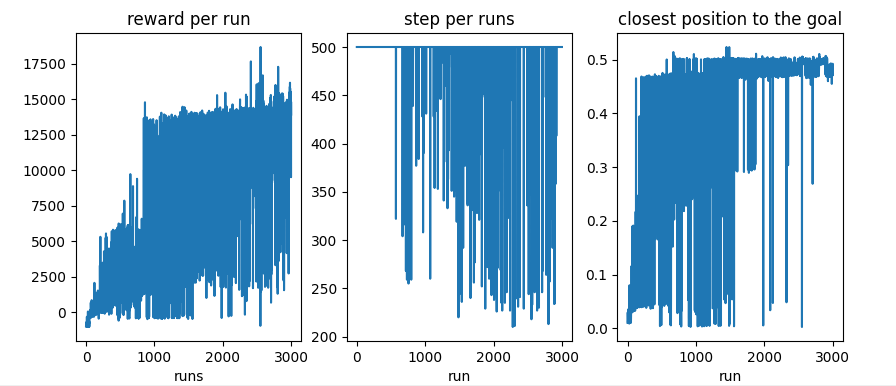
if state[0]>-0.7 and state[0]<-0.3:

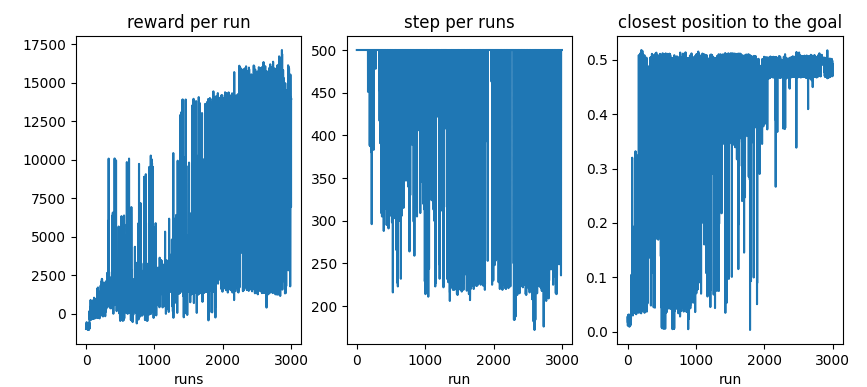
return 9\*(state[0]+0.3)

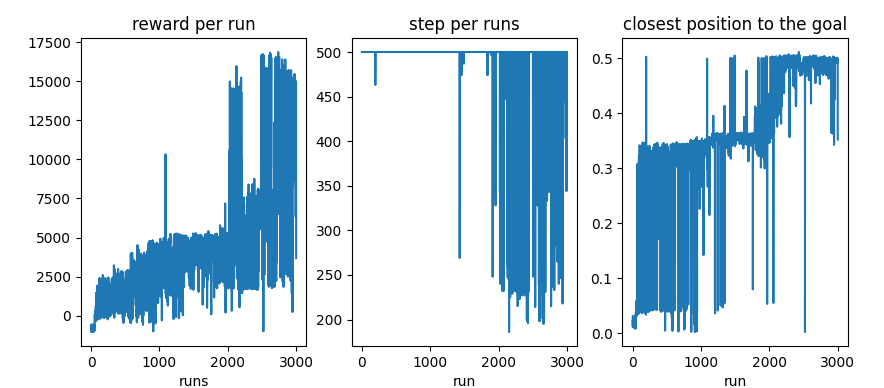
if state[0]>-0.2:

return (9\*(state[0]+0.3))\*\*2

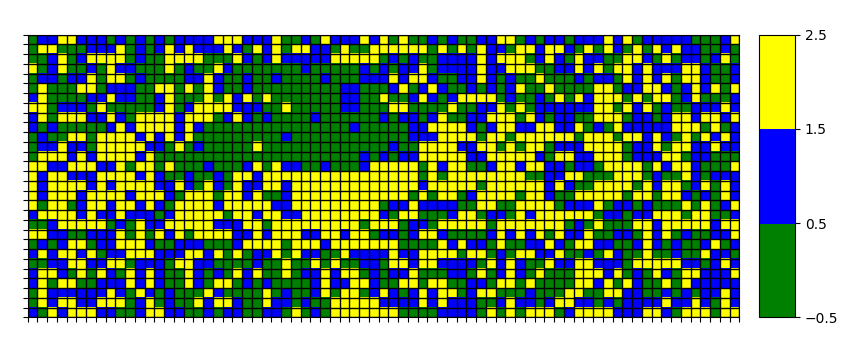
return 0



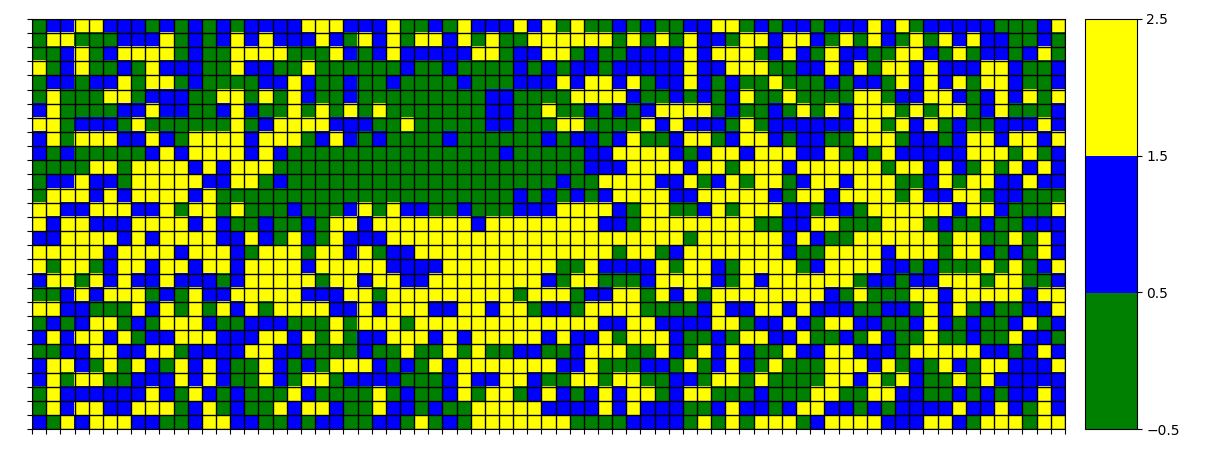




Run 1500



Run 2500



The case identified as optimal during the previous exploration. Like previous one but the exploration rate is now calculated by state

MAX\_RUNS=3500

MAXIMUM\_STEPS=500

INTERPOLATION=MAX\_RUNS/10

GAMMA = 0.95

LEARNING\_RATE = 0.25

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.75

self.exploration\_rate[state\_adj[1]][state\_adj[0]] \*= EXPLORATION\_DECAY self.exploration\_rate[state\_adj[1]][state\_adj[0]] = max(EXPLORATION\_MIN, self.exploration\_rate[state\_adj[1]][state\_adj[0]])

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

elif state[0]<=-0.7:

return ((state[0]+2))

elif state[0]>-0.7 and state[0]<=-0.3:

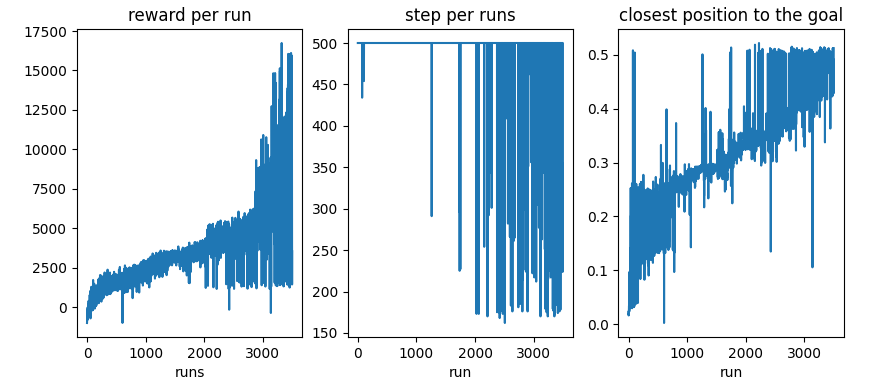
return 9\*(state[0]+0.3)

elif state[0]>-0.1:

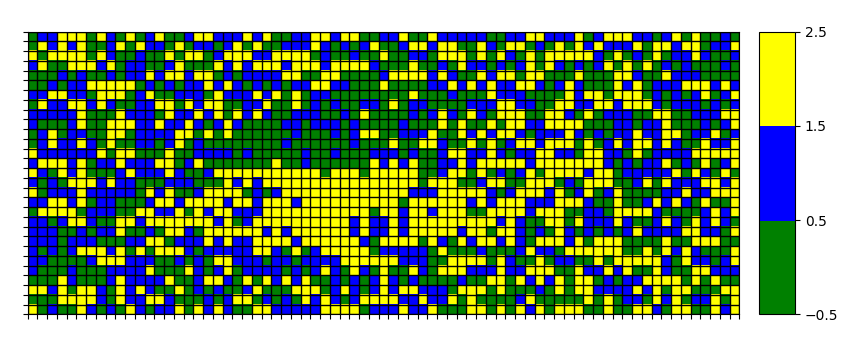
return (9\*(state[0]+0.3))\*\*2

else:

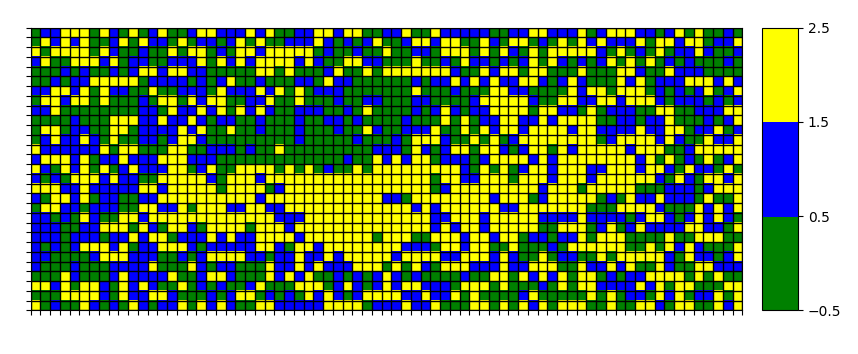
return 0



Run 1000



Run 2700



Same than first one in this section but without gap

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

if state[0]<-0.7:

return ((state[0]+0.7))

if state[0]>-0.7 and state[0]<-0.2:

return 9\*(state[0]+0.3)

if state[0]>-0.2:

return (9\*(state[0]+0.3))\*\*2

Same than second one in this section (exploration rate by state) but without gap

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

elif state[0]<=-0.7:

return ((state[0]+2))

elif state[0]>-0.7 and state[0]<=-0.3:

return 9\*(state[0]+0.3)

elif state[0]>-0.1:

return (9\*(state[0]+0.3))\*\*2

The same case where exploration is decreased by state, so the same level of exploration is achieved by all states and we also now penalty the number of steps, so the reward is lower as the time passes in the run.

MAX\_RUNS=2500

MAXIMUM\_STEPS=500

INTERPOLATION=MAX\_RUNS/10

GAMMA = 0.95

LEARNING\_RATE = 0.25

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.75

self.exploration\_rate[state\_adj[1]][state\_adj[0]] \*= EXPLORATION\_DECAY self.exploration\_rate[state\_adj[1]][state\_adj[0]] = max(EXPLORATION\_MIN, self.exploration\_rate[state\_adj[1]][state\_adj[0]])

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 250+(150\*((MAXIMUM\_STEPS-step)/100))

elif state[0]<-0.7:

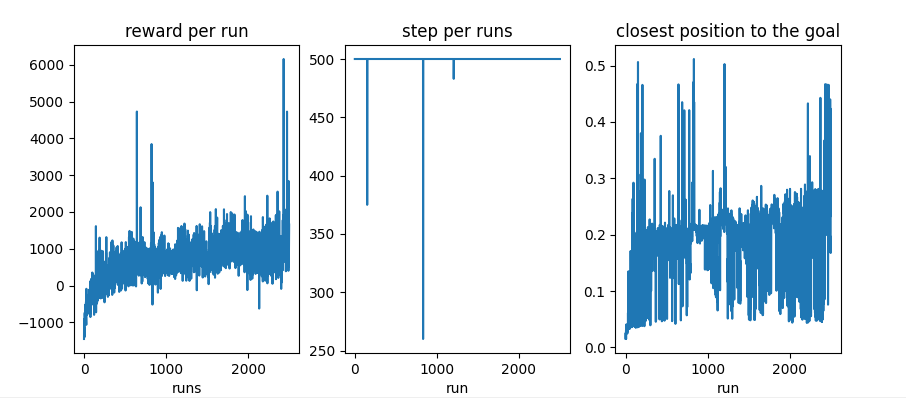
return ((state[0]+0.7))

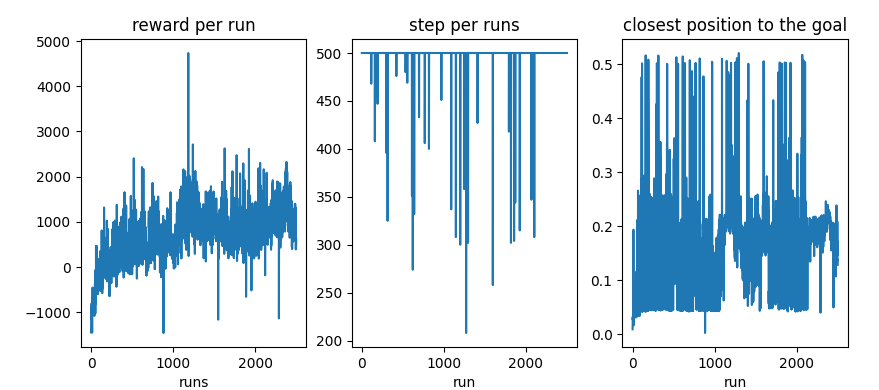
elif state[0]>=-0.7 and state[0]<=-0.3:

return 9\*(state[0]+0.2)

elif state[0]>-0.3:

return (4\*((MAXIMUM\_STEPS-step)/100)(state[0]+0.3))\*\*2





Same than the second one in this section (exploration rate by state) but without gap

MAX\_RUNS=3500

MAXIMUM\_STEPS=500

INTERPOLATION=MAX\_RUNS/10

ENV\_NAME = "MountainCar-v0"

GAMMA = 0.95

LEARNING\_RATE = 0.25

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.75

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

elif state[0]<=-0.7:

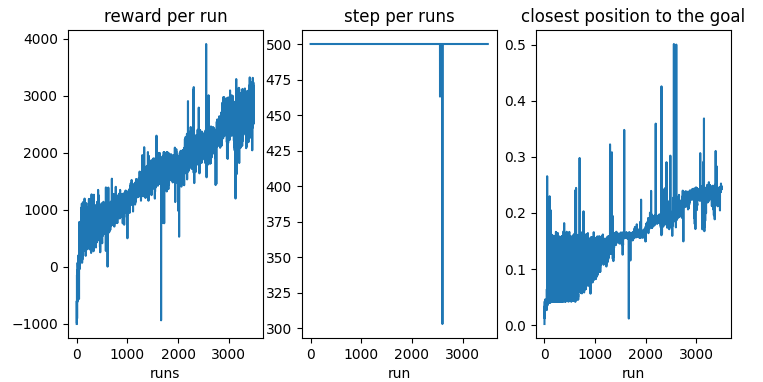
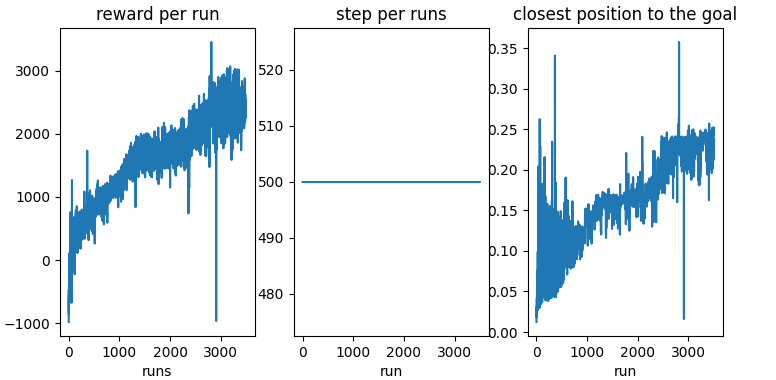
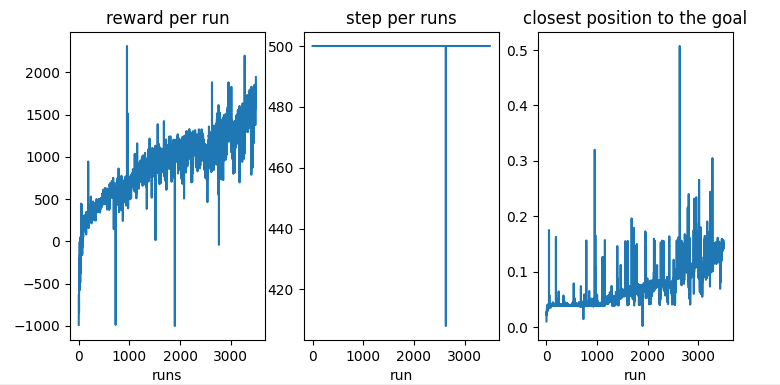
return ((state[0]+2))

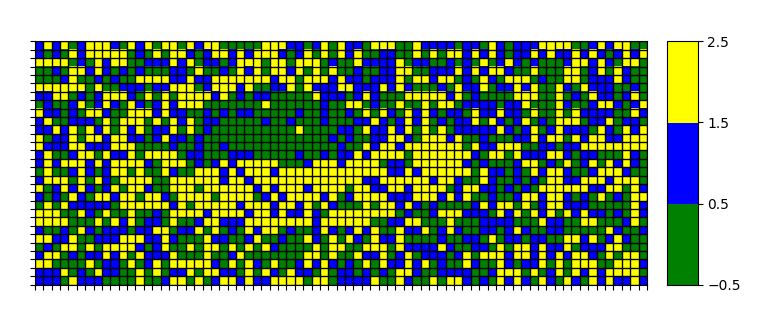
elif state[0]>-0.7 and state[0]<-0.2:

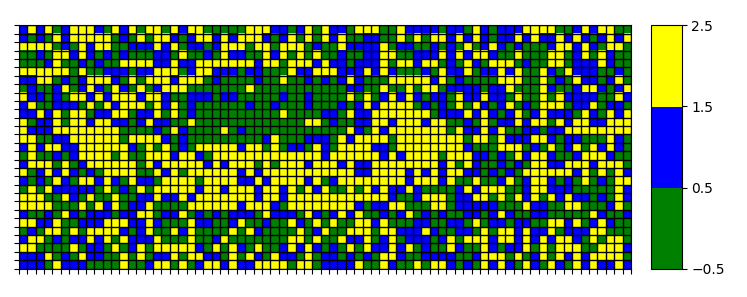
return 9\*(state[0]+0.3)

elif state[0]>=-0.2:

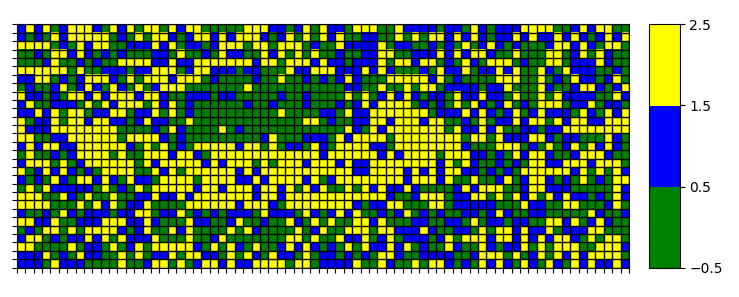
return (9\*(state[0]+0.3))\*\*2

Run 1750 

Run 3000 

Run 3500



Same than the first one in this section but without gap

MAX\_RUNS=3000

MAXIMUM\_STEPS=500

EXPLORATION\_STEPS\_PER\_STATE=100

GAMMA = 0.95

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.9995

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

elif state[0]<-0.7:

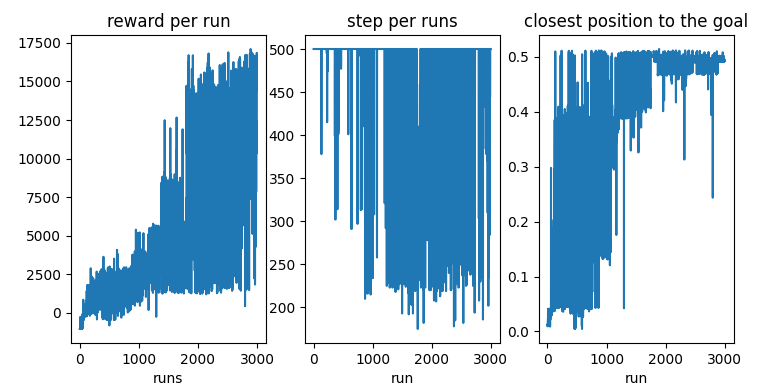
return ((state[0]+0.7))

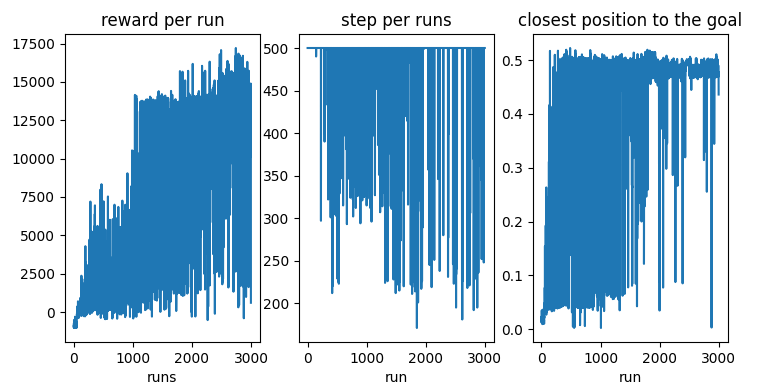
elif state[0]>-0.7 and state[0]<-0.2:

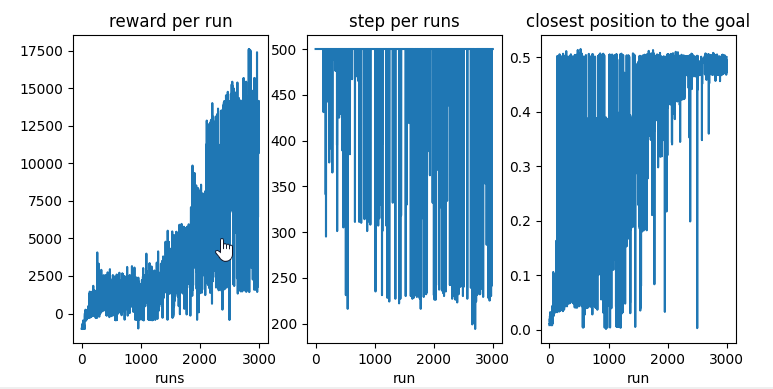
return 9\*(state[0]+0.3)

elif state[0]>=-0.2:

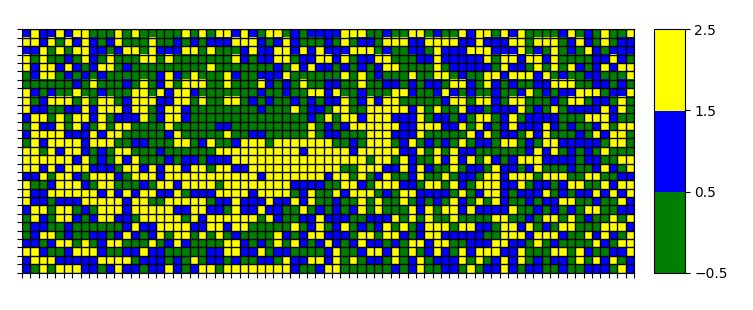
return (9\*(state[0]+0.3))\*\*2



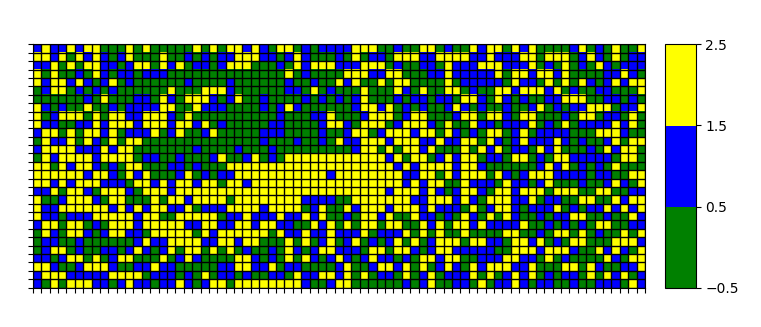




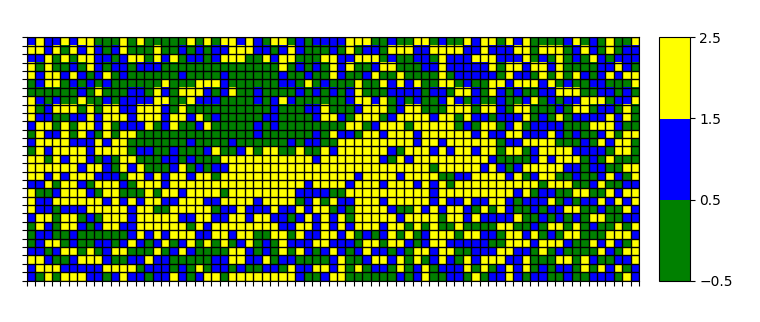
Run 600



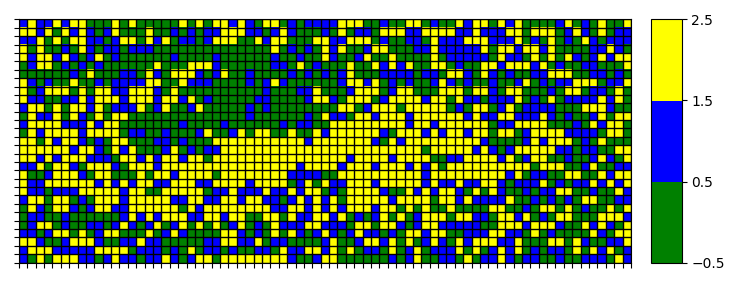
Run 1200



Run 2100



Run 3000



Same than the last one but with more exploration:

#It should be multiple of 10

MAX\_RUNS=2500

MAXIMUM\_STEPS=500

INTERPOLATION=MAX\_RUNS/10

ENV\_NAME = "MountainCar-v0"

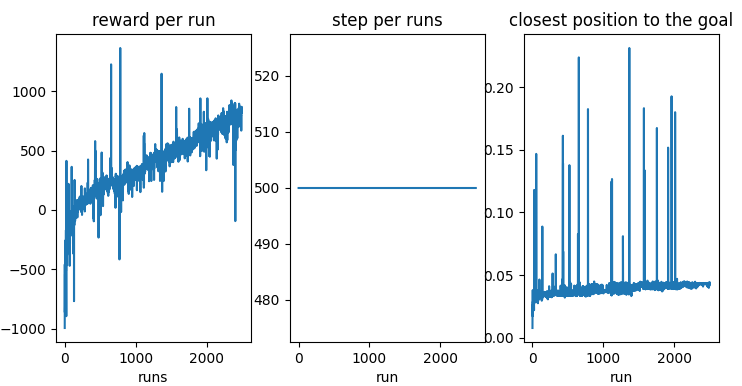
GAMMA = 0.95

LEARNING\_RATE = 0.25

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.9



Same than optimal one up to now but reswarding in terms of how high is the car in exponential way (more generic reward function) [AIM TO CONCLUDE que tiene que ser exponencial (y no es necesario indicar explicitamente que hay que evitar el valle a toda costa)).

GAMMA = 0.95

LEARNING\_RATE = 0.2

EXPLORATION\_MAX = 1.0

EXPLORATION\_MIN = 0.05

EXPLORATION\_DECAY = 0.99995

def get\_reward(state, step):

if state[0] >= 0.5:

print("Car has reached the goal")

return 500

elif state[0]>=0.1:

return 3\*\*(4\*(abs((state[0]\*10))))

