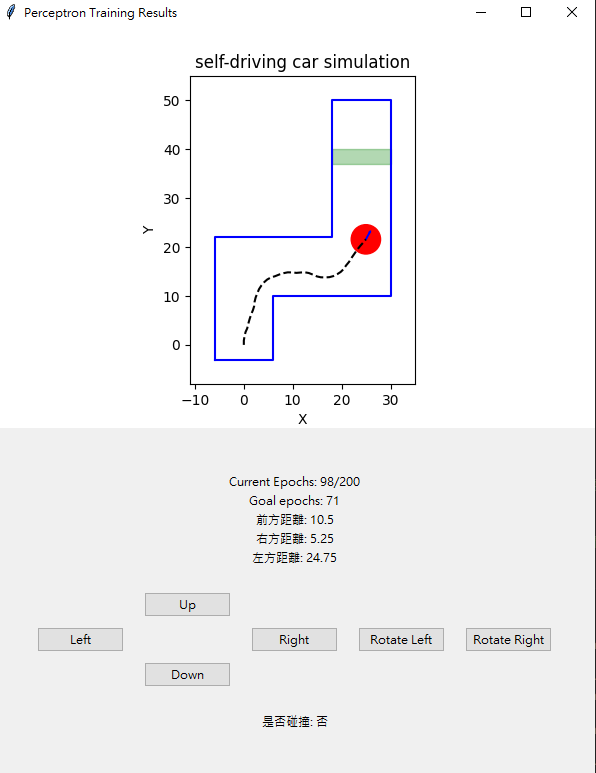
1. 程式介面說明

一開始於行程式會直接開始跑training，然後會持續顯示每次training的軌跡在GUI上

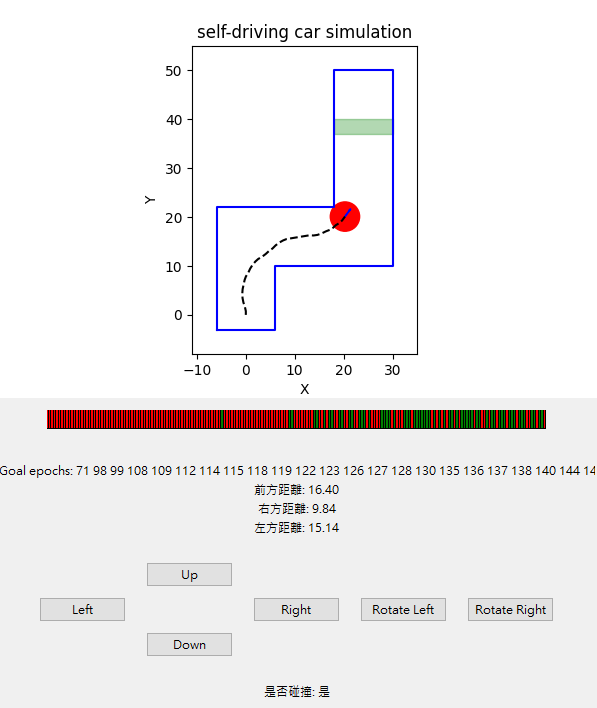
一張含有 文字, 螢幕擷取畫面, 圖表, 陳列 的圖片

自動產生的描述

當有任何的epochs抵達終點時，會更新Goal epochs到GUI上



全部訓練完成後(約5分鐘)會顯示訓練結果到GUI上，紅色為撞牆，綠色為到終點



下方按鈕為娛樂用，要按也可以按

1. 實驗結果

Q learning 參數

def \_\_init\_\_(self, degree\_per\_actions=4, state=5, learning\_rate=0.1, gamma=0.9, exploration\_rate=1.0, exploration\_decay=0.992):

State 定義

依據L-R的值做分配

一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Action定義

一格為4度，0=0度

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

Q table定義

Num\_state\*num\_actions

self.q\_table = np.zeros((self.num\_state, self.num\_actions))

1. 歸屬函數說明
2. def rew(self,distance):
3. return max(6.5-distance,0)\*\*2\*1000
4. def take\_action(self, action):
5. self.car.update\_state(self.convert\_action\_to\_angle(action))
6. next\_input=self.car.get\_distances()
7. next\_state = self.direction\_to\_state(next\_input)
8. reward=-5
9. if self.car.reach\_goal():
10. reward+=1000000
11. elif self.car.check\_collision():
12. reward=-200
13. p=(self.rew(next\_input[0])+self.rew(next\_input[1])+self.rew(next\_input[2]))
14. reward-=p #太貼牆會大幅減少reward
15. q=10\*(400 - MathTool.point\_to\_polygon\_distance(self.car.x,self.car.y,self.car.end\_area)\*\*2) #離終點越近 reward越高
16. reward+= q
17. #print("reward:",reward,"wall:",p,"distance:",q)
18. return next\_state, reward

當抵達終點，給予一個極大的正面reward

撞到牆壁，給予一個小的負面reward

接著判斷三個方向的rew值，判斷依據是是否離牆很近，如果距離<6.5，則給予蠻大的負面reward，離>6.5則無事發生，然後把三個方向的rew值加總



rew function的圖表

再來判斷車輛到終點的距離，距離<20有中幅的正面reward，>20有中幅的負面reward

一張含有 文字, 行, 繪圖, 圖表 的圖片

自動產生的描述  
4. 分析

exploration\_decay不要設太小，也要給後續的路留一些探索的機會

Q table不要設太大，否則跑1000epoch也跑不完

用離牆距離判斷應該不是最佳解，不然車頭過了他就會認為離牆很遠很安全，然後尾巴就撞車了