Algorithm: **Q-Learning**

```
EPSILON = 0.7
GAMMA = 0.99
LEARNING_RATE = 0.01

train_episode = 5000
step_limit = 2000
```

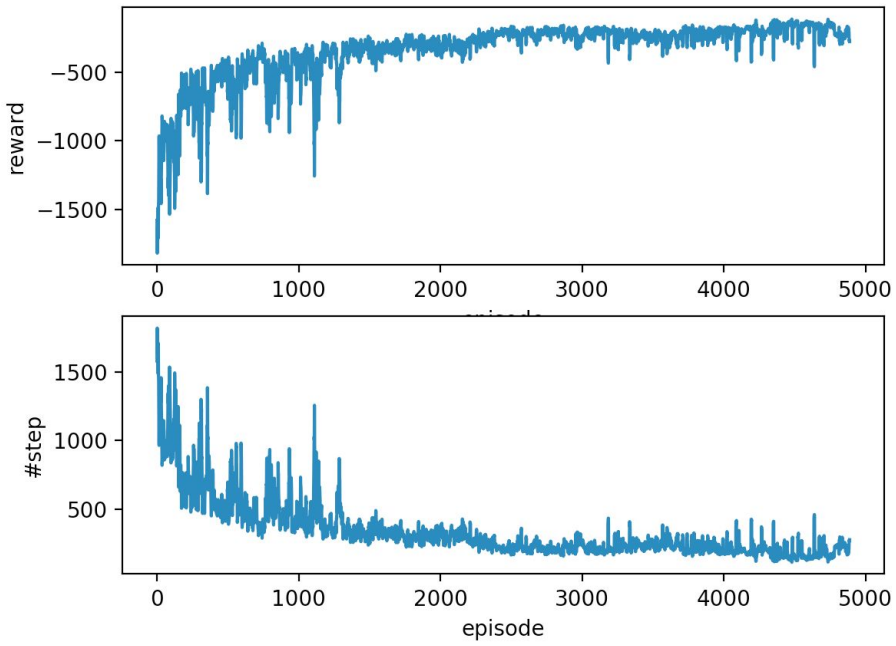
epsilon 會隨著episode的演進慢慢成長到1(線性成長)，也就是一開始車子會有比較高的機率隨機動作(explore) 等到train越接近結束亦即episode越大，model就會傾向選擇q table中value最大的動作(greedy)

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Q-learning會根據現在的state選擇對應action
其實我原本只考慮position作為state但是因為q table更新錯了所以失敗

現在我的state是一組(posision, velocity) set，各切成10等分，所以有100 states
每個state有三個action space = [0, 1, 2]

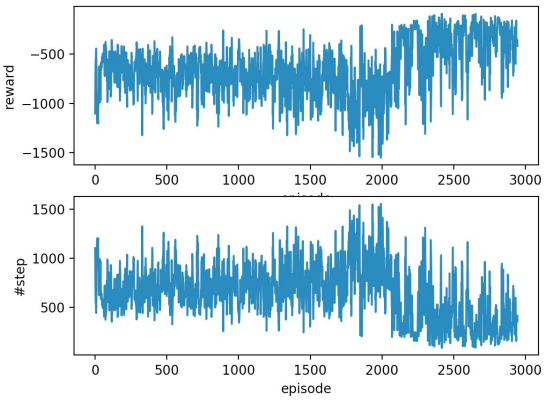
fig1: moving average(slide window size=5)

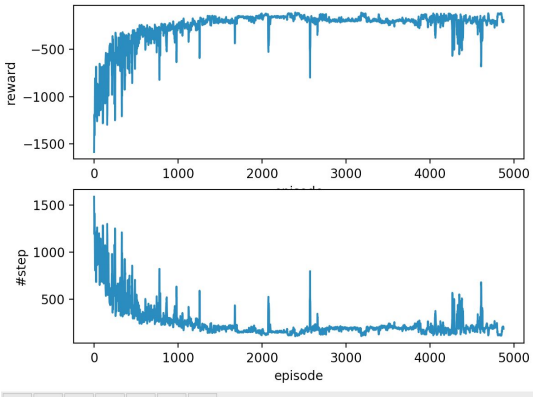


可以看到在train過程中，reward預設為每次作一個動作就-1 所以這個表示越快到達終點，reward越高，而兩張圖是對稱的，結果可以看到大約1250步之後就比較穩定了

fig2: 調整不同state size

左邊：5*5　　右邊: 10*10





可以看到state太少的時候train不太起來，雖然還是有在2000 episode時有提升，但整體過程都很崎嶇不穩定而且train的時間也比較久

當state數提升10*10的時候收斂的比較快 training效果也有比較明顯的提升，而比較快到達終點，所需時間較短，結果卻沒有比15*15 train出來差

結論是state數越多，效果應該會越好，因為精細度越高，但是q table位置也變多在探索期間必須花更多時間完成表格，但一旦train完成後反而可以更快到終點

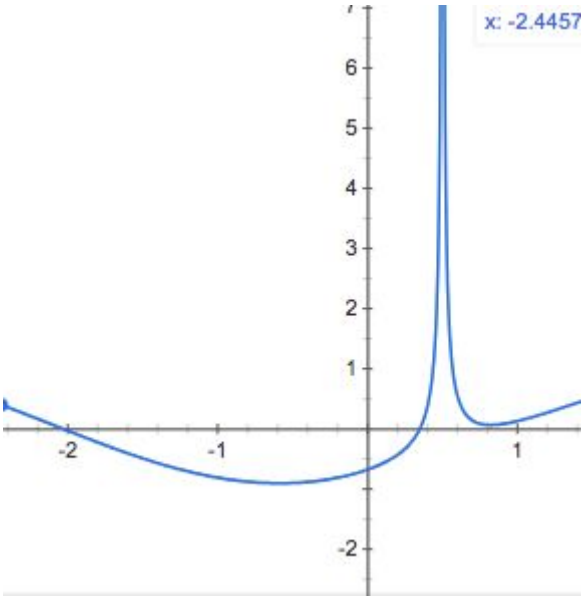
增加state對於model幫助有上限，因為state太多，training時間就會太久，效果卻不會差太多

調整reward function: 我覺得reward 可以考慮實際的地理位置(cos function)和離goal的距離一起的效果

- cosine: 反映在山上的高度(越高越好，因為旗子在高處，且需要利用擺盪的力量爬山)
- distance: 反映離旗子距離(越近越好)

因此設計這個reward function:

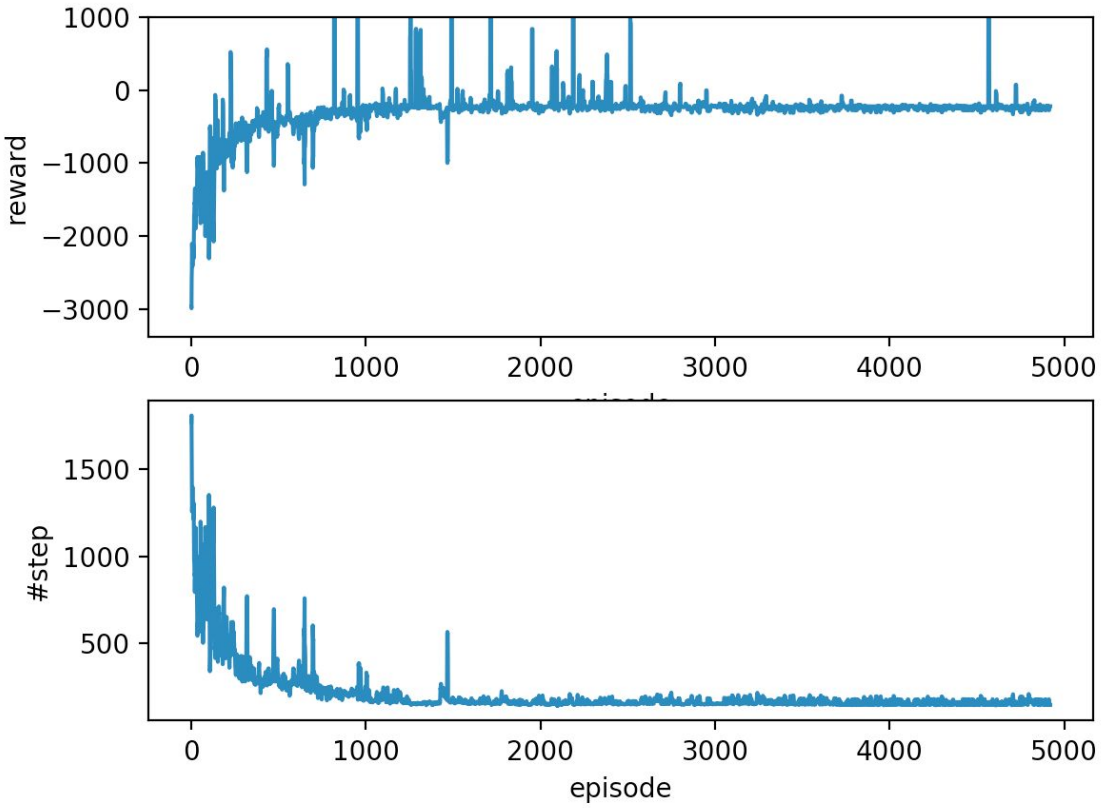
$$reward = -cos(position + 0.5) + 0.1/|position - 0.5|$$



如圖

顯現出在山坡上的reward會較高，而且旗子在右邊，所以右邊的reward又比左邊高(右邊斜率較大)

fig3: use my reward function:



比原本的單純reward更早收斂也更stable(by #step)，所以這個reward function應該是有幫助的

---

1. What kind of RL algorithms did you use? value-based, policy-based, model-based? why? (10%)
   - Q-learning, value-based, 因為我maintain一張q table，並用q value作為動作準則，而非train出真正理解環境的policy和model, 單純random或greedy
2. This algorithms is off-policy or on-policy? why? (10%)
   - off-policy, 因為估計的策略和生成樣本的策略不一定相同(估計policy永遠是max action)
3. How does your algorithm solve the correlation problem in the same MDP? (10%)
   - q table中的值記錄了train過程中每個state變換時該選那個action會最好，對應到MDP，每個qtable index就是一個MDP的state, q value可以反映MDP trainsition機率，這個機率就會和一串的狀態和動作有關(看gamma的設定)
   - 這樣可以間接瞭解state, action之間的correlation