AI Assignment 2

2016147540 컴퓨터 과학과 안범진

목차

1. 과제의 내용과 목적

이번 과제는 "Grab the Coin!" 이라는 이름 그대로 게임 화면 상단에서 무작위로 떨어지는 코인과 시간 아이템을 먹는 agent를 Q-learning을 통해 학습시키는 것이었다.

Grab the Coin!

간단하게 게임의 룰을 설명하자면 다음과 같다.

- 떨어지는 코인을 먹으면 +500점, 먹지 못하면 -150점
- 시간 아이템을 먹으면 counter(남은 시간) +100 (점수에는 영향을 주지 않는다)
- 시간 아이템은 counter가 500보다 작을 때 나오고 점수가 300,000점을 넘었을 때는 더 이상 나오지 않는다.
- counter는 처음에 200으로 시작한다.
- 아이템이 떨어지는 위치는 0~11의 index로 표현하고 바스켓이 갈 수 있는 위치는 0~8의 index로 표현한다.
- 아이템의 정보는 (id, x, y)로 구성되어 있다. id=1 score item, id=2 time item
- 바스켓은 한번에 한칸씩 좌우로 움직이거나 제자리에 멈춰 있을 수 있다.

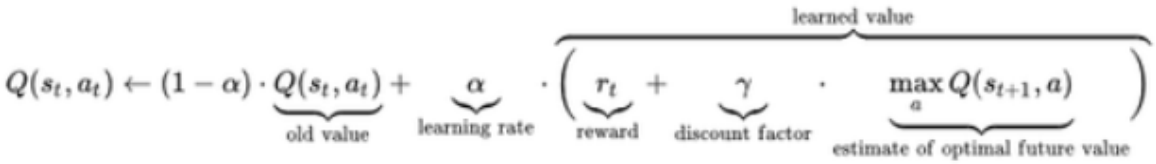
  위 룰을 따르며 가장 큰 스코어를 얻는 것이 이 게임의 궁극적인 목적이다.

2. Q-learning에 대하여

이번 과제에서 쓰인 Q-learning에 대해서 알아 보자. Q-learning은 Machine learning에서 쓰이는 강화학습 방법중 하나이다. 특별한 조건이 필요하지 않고 상태 변화와 보상을 통해 최적의 행동을 찾고, 개선해 나갈 수 있다.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Q 값을 업데이트 하는 방법은 위와 같다. 아래와 같이 쉽게 다시 보자.



reward에 가장 최선의 예측값 * discount factor을 한 값을 더해준 후 learning rate와 다시 한번 곱해준다. 그리고 이 값과 (이전 값) * (1-learning rate) 값을 더해준다. Learning rate가 클 수록 미래지향적이고 작을 수록 과거의 값에 더 중요도를 둔다. discounter factor는 클수록 빨리 리워드를 얻는걸 중요하게 여긴다.

이번 과제에서는 basket이 왼쪽, 오른쪽 또는 정지해 있을 수 있으며 아이템은 게임 상단 화면에서 아래로 떨어진다. 각각의 상황에서 최적의 transition을 찾아 내야한다. 그리고 reward함수도 적절히 짜서 Q-learning이 성공적으로 이뤄지게 해야한다. learning rate, discounter factor, epsilon등 Q-learning에 필요한 변수를 적절히 정해보자.

3. state-function, reward-function

I. state-function

이번 과제에서 가장 크게 변한 코드를 꼽자면 이 get_state함수일 것이다. 코드를 한번 보자.

```
def get_state(counter, score, game_info):
    basket_location, item_location = game_info
    """
    FILL HERE!
    you can
    (and must) change the return value.
    """
    itemnum = len(item_location) #number of item
    check=False #bool variable for item check
    check2=False #bool varialbe for item check
    movement=0 #result of function
    time=0 #find index of time item
    item=0 #find index of item
    for j in range (0, itemnum): #find time item function
        name=item_location[j][0]
        if(name==2):
            time=j
            check2=True
            break
    if(check2==True):#There is time item
        check2=True
    else:#There is not time item
        for i in range (0, itemnum):
            x=round(item_location[i][1]*7/11)
            y=10-item_location[i][2]
            if(y-abs(basket_location-x) > 0):
                item=i
                check=True #Find can go

                break
```

```
    if(check==False and check2 == False): #Can't find can go
        if(basket_location>4):
            movement=1
        if(basket_location<4):
            movement=2
        if(basket_location==4):
            movement=0
    if(check==False and check2==True): #go to time item
        if(basket_location>round(item_location[time][1]*7/11)):
            movement=1
        if(basket_location<round(item_location[time][1]*7/11)):
            movement=2
        if(basket_location==round(item_location[time][1]*7/11)):
            movement=0
    if(check==True and check2==False): #go to score item
        if(basket_location>round(item_location[item][1]*7/11)):
           movement=1
        if(basket_location<round(item_location[item][1]*7/11)):
           movement=2
        if(basket_location==round(item_location[item][1]*7/11)):
           movement=0

    return movement
```

위와같이 get_state를 정의하기 까지엔 많은 시행착오가 있었다. 처음 코드를 짰을때는 주어진 코드에서 return값이 basket_location 으로 되어있어서 "아, 리턴값을 지금 가야하는 basket의 위치 즉 0~8으로 줘야 하는구나" 라고 생각을 하고 해서 학습을 시켜도 점수가 나아지지 않고 평균 점수는 10000점대에 약간 못미치는 점수를 얻게 됐다.

알고리즘을 나름 완벽하게 짰다고 생각하는데도 불구하고 나아지지 않는 점수가 너무 이상해 get_state를 벗어나 코드 전체를 보기 시작했다. print 함수를 통해 Q 변수와 A 변수에 어떤 값이 들어가 있는지 확인하는 과정을 거쳐 get_state 의 return 값인 basket_location은 0, 1, 2 즉 현재 상태에서 어디로 가야하는지를 리턴해 줘야 한다는 결론에 도달하게 됐다.

위 코드에 대해서 설명을 하자면 일단 game_info에서 가져온 item_location의 길이를 구한다. item_location의 길이가 의미하는 것은 현재 떨어지고 있는 아이템을 의미한다. index가 0에 가까울 수록 아이템의 y좌표가 9에 가깝다는 소리이다.

```
for j in range (0, itemnum): #find time item function
        name=item_location[j][0]
        if(name==2):
            time=j
            check2=True
            break
```

위 부분은 현재 떨어지고 있는 아이템중 시간 아이템이 있나 확인하는 코드이다. 만약 떨어지고 있는 시간 아이템이 있다면 점수 아이템을 뒤로한채 시간아이템 밑에서 기다리게 했다. 시간만 충분하다면 30만점까지는 무조건 갈 수 있다는 생각으로 했다.

```
else:#There is not time item
       for i in range (0, itemnum):
           x=round(item_location[i][1]*7/11)
           y=10-item_location[i][2]
           if(y-abs(basket_location-x) > 0):
               item=i
               check=True #Find can go
               break
```

다음으로, 위 부분은 만약 현재 떨어지고 있는 시간 아이템이 없을 때 먹을 수 있는 점수 아이템으로 가라는 코드이다. 아이템 중에서 가장 y좌표가 큰 것 부터 본다. 현재 basket_location에서 먹을 수 있는지 x, y좌표를 이용해 판단하고 갈 수 있다면 그 아이템으로 가고 갈 수 없다면 다음 아이템을 먹을 수 있는지 검사한다.

나머지 코드는 check, check2, 현재 basket_location, 먹으러 갈 item의 위치에 따라 가는 방향을 정해주는 부분이다.

II. reward-function

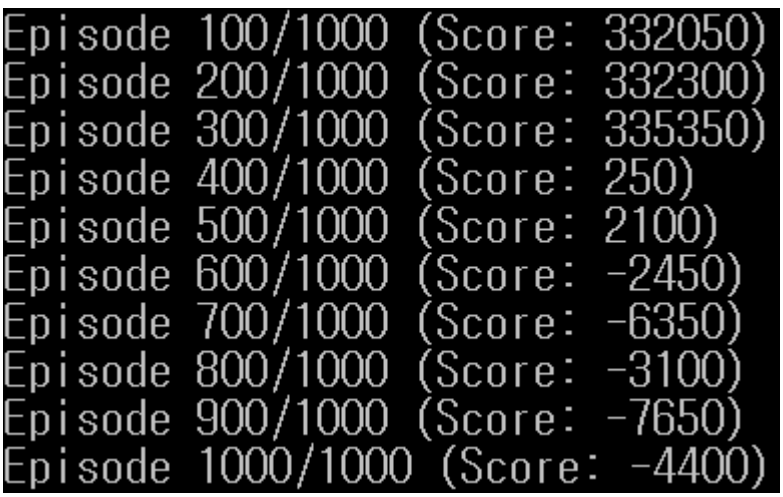
A. 게임이 끝났을 때 reward = -500

- 331800, 337400, 335250, 329100, 328400, 336550, 334800, 335150, 340800, 324450
- 평균: 333,370

B. 게임이 끝났을 때 reward = -1000

- 318000, 316250, 320100, 322750, 248750, 328400, 280700, (11850), 329250, 332900, 330750
- 평균: 312,785

C. 게임이 끝났을 때 reward = -5000



- 러닝을 시킬때 부터 이상하다.
- -3650, -1150, -8200, -3750, -2850, -9400, -6300, -4400, -5450, -2850
- 평균: -4800

D. Original code (게임이 끝났을 때 reward 변화 없음)

- 333500, 335050, 323650, 325600, 345300, 324700, 81600, 324950, 334450, 330600
- 평균: 305,940

A, B, C, D 중에서 A를 채택하자.

A-1 Counter < 200 일 때 counter에 가중치 3

- 331550, 317400, 342450, 329850, 329450, 339200, 336750, 117950, 336450, 332150
- 평균: 311,320

A-2 Counter < 200 일 때 counter에 가중치 10

- 331350, 339500, 335000, 335650, 338400, 330200, 335100, 341700, 323900, 340850
- 평균: 335,165

A-3 Counter < 200 일 때 counter에 가중치 30

- 336950, 320600, 337150, 337850, 337950, 329050, 344450, 342050, 324350, 16650
- 평균: 302,705

A-4 Counter < 200 일 때 counter에 가중치 100

- 341350, 335750, 337350, 336000, 341150, 338850, 334650, 338300, 328300, 337700
- 평균: 336,940

A-5 Counter < 200 일 댸 counter에 가중치 500

- 194100, 285200, 332500, 328800, 14050, 323400, 319600, 336000, 203650, 129450
- 평균: 246,675

counter에 대한 가중치가 너무 크지만 않으면 모두 비슷한 결과를 얻을 수 있다는 결론을 내렸다. 그래서 적당한 값을 선택해 줬다 > 최대 평균 > 가중치 = 10 채택.

4. policy-function, Q-score

policy 함수와 q_learning 함수안쪽 코드는 건드리지 않았다. 따라서 이 두 함수가 어떻게 작동하는지 설명하겠다.

I. policy-function

```
def make_policy(Q, epsilon, nA):
    def policy_fn(observation):
        A = np.ones(nA, dtype=float) * epsilon / nA
        best_action = np.argmax(Q[observation])
        A[best_action] += (1.0 - epsilon)
        return A
    return policy_fn
```

- make_policy 함수는 return 값으로 policy_fn 함수를 실행 시킨다
- observation에는 get_state함수에서 return 해주는 값의 갯수, 즉 (0, 1, 2) 3개중 한개가 들어간다.
- A를 [epsilon/nA, epsilon/nA, epsilon/nA] 리스트로 만들어준다.
- Q[observation] 리스트 안에 가장 큰 수의 index를 best_action에 넣어준다. 이는 곧 내가 움직이게 될 방향이 된다.

- A[best_action] 에 1-epsilon을 더해줘서 A의 리스트안 값의 총합이 1이 되게 만들어 준다.

  요약: 지금 보는 Q 값중에서 가장 큰 값의 index를 best action 으로 놓고 A를 [epsilon/3, 1-2epsilon/3, epsilon/3] 으로 만들어 준다. 이 A 리스트를 리턴해 준다.

Q-score

```
best_next_action = np.argmax(Q[next_state])
td_target = reward + discount_factor * Q[next_state][best_next_action]
td_delta = td_target - Q[state][action]
Q[state][action] += alpha * td_delta
```

- 다음으로 가야하는 state가 next_state에 들어가 있고 next_state에서의 best_next_action을 Q값을 통해 찾는다. Q[next_state]에 들어있는 가장 큰 값의 index를 best_next_action에 넣는다
- td_target에 reward + 0.5 * Q[next_state][best_next_action] 을 넣어준다. (discount_factor의 default 값 = 0.5)
- td_delta에 td_target에서 Q[현재 상태][행동] 값을 빼준다.
- Q[현재상태][행동]에 알파(learning rate)*td_delta 값을 넣어준다.

  요약: 다음 상태의 Q 값에서 가장 큰 값의 index를 best_next_action에 넣어주고 Q값을 계산해주고 업데이트 시켜준다.

5. Learning parameter

I. learning rate

앞서 설명했듯이 learning rate가 크다면 미래지향적이고 작다면 과거의 데이터에 더 중요도를 둔다. epsilon과 다르게 값이 커야 좋은지 작아야 좋은지 감이 오지 않아 처음에는 아주 크게 0.99로 학습을 시켜봤다. 그랬더니 바스켓이 전혀 움직이지 않고 epsilon에 의해 랜덤으로 움직일 때만 움직이는걸 볼 수 있었다. 그리고 default값이 0.1인 걸 보아 이 정도가 적당한 것 같아 여러 수를 대입해보다가 0.01이 가장 score가 잘 나오는 것 같아서 0.01을 채택했다.

II. epsilon

action을 선택하는데 있어 Q값 중 가장 큰 값을 선택한다. 이 때 일정확률(=epsilon)로 다른 랜덤한 action을 선택하도록 해주는 변수이다. 하지만 학습이 많이 진행된 이후에도 초기의 큰 eplsilon값이 유지된다면 큰 문제가 아닐 수 없다. 충분한 학습을 통해 최적의 행동을 학습했는데 그걸 무시하고 랜덤으로 행동할 수는 없지 않는가? 그래서 딥 마인드의 경우에는 eplsilon값을 학습에 거쳐 점점 줄여주는 방법을 채택했다. 학습 초기에는 랜덤한 action을 할 확률이 높고 학습 말기에는 학습한 최적의 행동을 할 확률이 높아진다.

이번 과제에서도 epsilon-greedy로 짜여진 코드를 바꿀 수 있게 해주셨는데 이걸 decaying epsilon-greedy로 바꿨다면 더 좋은 학습 결과를 얻을 수 있었을 것이다. 하지만 능력부족으로 인해 여기까지는 달성하지 못했다. 0.01을 채택했다.

6. Result of my agent

위와 같은 분석을 통해 내 코드에 가장 알맞는 변수를 이용해 러닝을 시켰다

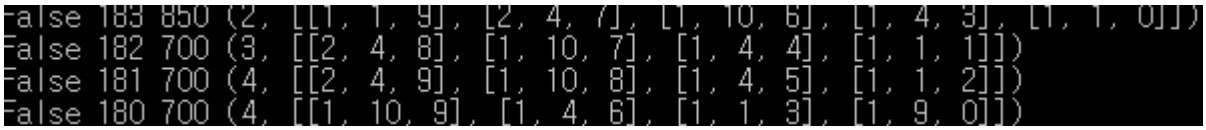
- python q_learning.py -n 10000 -e 0.01 -lr 0.01
- 이후 30번을 실행시켜 통계를 냈다.

1회	**23650**
2회	337100
3회	344550
4회	341150
5회	337100
6회	14450
7회	336700
8회	156000
9회	336100
10회	348550
11회	333200
12회	335250
13회	334100
14회	342600
15회	336950
16회	341600
17회	337300
18회	326950
19회	325750
20회	329000
21회	334050
22회	339450
23회	338400
24회	335650
25회	341300
26회	326750
27회	333600
28회	328350

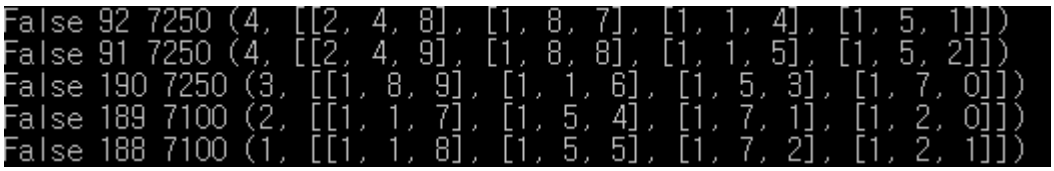
1회	23650
29회	332100
30회	324900
평균	308,420

위 표를 보면 알 수 있듯이 대부분이 30만점을 모두 넘었다. 30만점까지 시간 아이템을 잘 먹어가며 점수를 쌓고 30만점 이후로는 시간 아이템이 나오지 않으니 코인만 먹다가 게임이 끝나는 것이다. 하지만 30회중에서 2회가 10만점을 넘지 못했고 1회가 20만점을 넘지 못했다.

알고리즘 상의 문제인지 학습상의 문제인지 랜덤 action상의 문제인지는 확신이 서지 않지만 점수가 너무 낮은 게임일 때 cmd창에 뜨는 결과를 분석해 본 결과 아래와 같았다.



게임이 시작하고 나온 처음 timer를 먹지 못하고

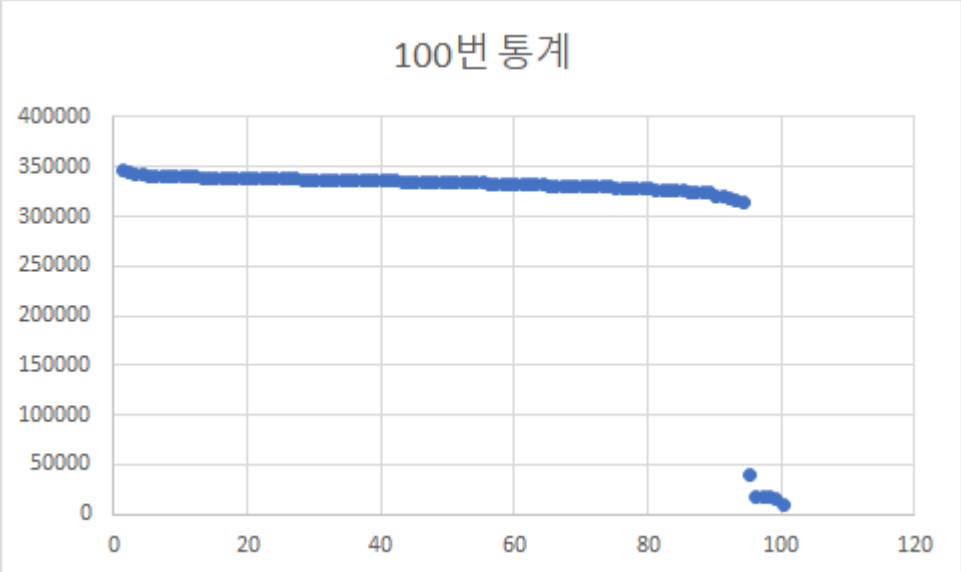


그 다음에 나온 timer는 먹고 이 뒤로 게임이 끝날 때 까지 timer 아이템이 한번도 나오지 않은 경우였다. 물론 처음 timer를 놓친것이 문제지만 2개중 한개를 놓쳤다고 게임이 끝나는, 너무 극단적인 게임이었다고 할 수 있다.

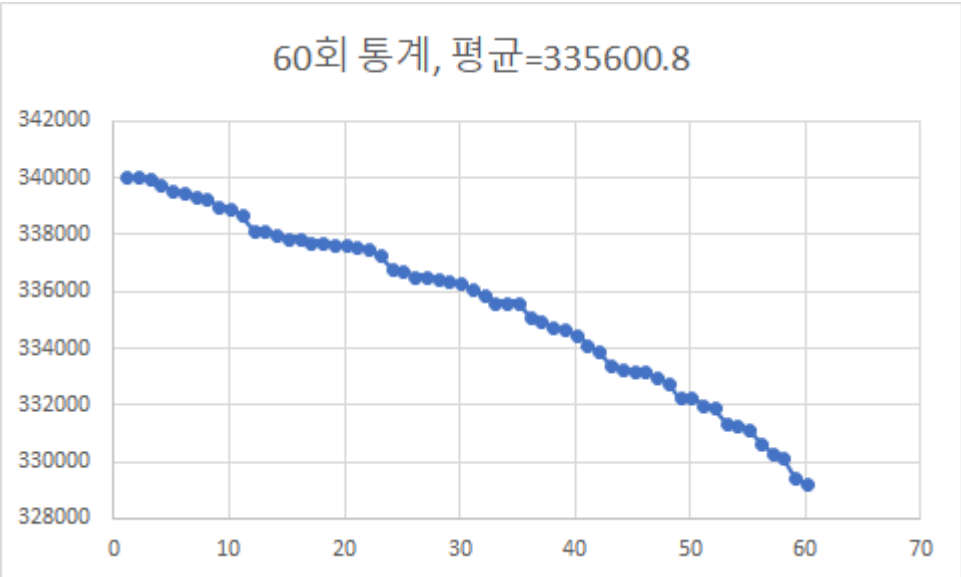
이런 극단적인 경우가 아니라면 거의 모든 경우 30만점을 넘어가는, 그니까 타이머가 나온다면 게임을 계속해서 할 수있는 그런 학습이 진행되었다고 할 수 있다. 대부분의 경우 타이머가 나오는 30만점 전까지 계속해서 400~500대의 counter를 유지해 가며 게임이 진행된다.

조금 더 자세하게 분석을 해보자

앞서 한 30번의 실행 횟수로는 제대로 된 평균을 내지 못한 것 같고 an.yonsei.ac.kr 페이지에 올라와 있는 agent 평균 내는 방법이 총 500번 실행해서 하위 100번과 상위 100번의 결과를 제외한 중앙 300번의 평균을 낸다고 돼있어서, 총 100번을 실행해서 하위 20번과 상위 20번을 제외한 중앙 60번의 평균을 냈다.

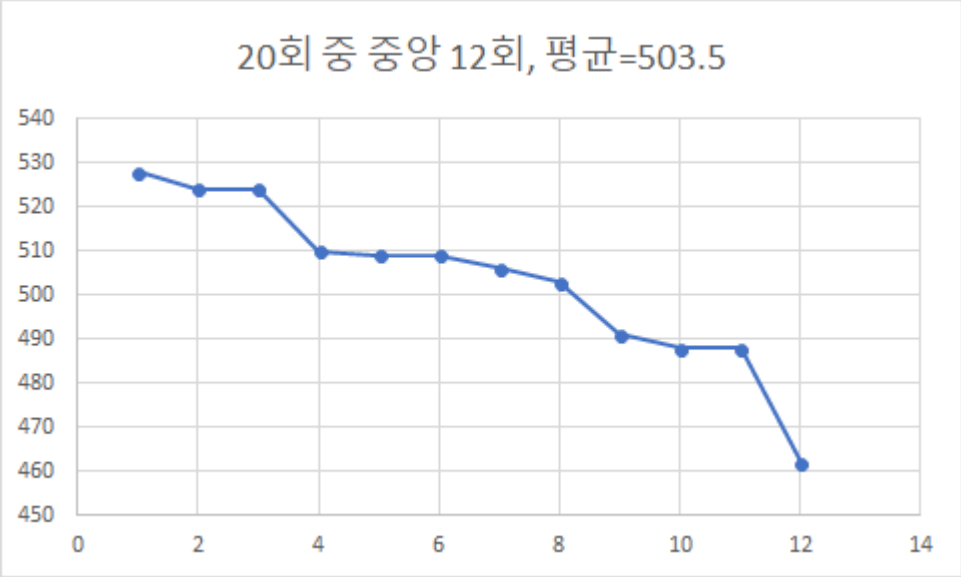


100번의 결과는 위 그림과 같이 분포해 있었다. 30만점 이상이 94회, 30만점 이하가 6회였다. 30만점이 넘거나, 30만점을 넘지 못하면 10만점도 못하는 이 두 그룹으로 나뉘었다. 100회는 일반화 하기 충분한 실행 횟수라고 생각해 6% 정도를 초반에 타이머를 잘 먹지 못하는 비율이라고 생각하겠다.



중앙 60번의 통계를 보면 최대값 340050부터 329250까지 고르게 분포해 있음을 볼 수 있다. 평균값은 335600점으로 30만점이 넘은 이후로 35600점을 더 획득했다. 30만점까지 도달 했다는 소리는 도달 전까지 타이머를 꾸준히 잘 먹어왔다는 소리고 이는 30만점에 도달 했을 때 큰 counter값을 가지고 있다고 생각 할 수 있다. 500정도로 가정해 볼 수 있겠지만 보다 정확한 분석을 위해 통계를 내보자.

- 100회 중 중앙 60회 평균은 335600



30만점에 도달했을 때 남은 시간평균이 503.5가 나왔다. 그러므로 503.5counter동안 평균적으로 35600점을 획득했다고 할 수 있다.

- 70.71점/counter라는 식을 얻을 수 있다.

7. Reference

- http://egloos.zum.com/incredible/v/7392473 Q learning에 대하여 참고
- http://egloos.zum.com/incredible/v/7392473 learning rate, discount factor, eplsilon, decaying epsilon greedy에 대하여 참고