Blackjack With Monte-Carlo Control

학과 : 컴퓨터공학과
학번 : 12161532
이름 : 김난영

<Index>

1. 게임 설명
2. 게임 규칙
3. 과제 요구 조건
4. 작성한 코드
5. 추가 과제 요구 조건
6. 결과 및 고찰

[1. 게임 설명]

플레이어와 딜러는 각각 두 장의 카드를 받음
딜러는 카드 한 장을 플레이어에게 공개

<플레이어>
- 플레이어는 카드를 더 받거나(Hit) 받지 않을 수 있음(Stick).
- 여러번 Hit 가능.
- 플레이어가 더 이상 카드를 받지 않으면 딜러의 순서가 됨.

<딜러>
- 플레이어가 Stick한 후부터 카드의 합이 17 미만일 때 항상 추가 카드를 뽑음(Hit)
- 카드의 합이 17 이상일 때 멈춤 (Stick)

[2. 게임 규칙]

- 하나의 카드 덱(조커를 제외한 52장의 카드)을 사용
- 카드의 합을 계산할 시 A = 1 or 11, J,Q,K = 10으로 계산
- 플레이어는 Hit 또는 Stick만을 할 수 있음
- 카드의 합이 21 이하일 경우, 딜러와 플레이어 중 숫자가 더 높은 쪽이 승리.
- 카드의 합이 21 초과한 딜러/플레이어는 패배
- 카드의 합이 같을 경우 무승부

본 프로그램에서는 최적 Policy를 찾기 위해 Monte-Carlo Control with Exploring Start[1] 알고리즘을 사용합니다. 최적 Policy를 찾기 위해, 본 프로그램은 다음과 같은 변수로 구성된 State로부터 탐색을 수행합니다.

- 플레이어 카드의 합 : 12 ~ 21 사이의 정수
- 딜러가 보여주는 카드의 숫자 : 1 ~ 10 사이의 정수
- 플레이어가 현재 사용 가능한 Ace의 유무 : True / False

위와 같이 구성된 State S에서 Action A를 선택했을 때의 기대 리턴(게임이 끝났을 때 얻을 것으로 기대되는 보상의 합)은 Q(S,A)입니다. 많은 에피소드를 경험할수록 Agent는 Optimal Q(S,A)에 가까운 값을 학습하게 되며, 이로부터 최적 Policy를 찾을 수 있습니다.

[3. 과제 요구 조건]

<목표>
위 코드로부터 최적 Policy를 학습시킨 Agent를 사용하여, 블랙잭 게임을 시뮬레이션한 뒤 승률을 계산.

<초기 설정>
- 플레이어는 1,000,000번의 에피소드로부터 최적 Policy를 학습
- 플레이어의 초기 자금 : 10,000 달러
- 플레이어는 게임 참가 시 10 달러를 지불, 결과에 따라 금액을 획득
  - 승리 시 : 20 달러 획득
  - 무승부 시 : 10 달러 획득
  - 패배 시 : 0 달러 획득

<요구사항>
- 플레이어와 딜러가 1,000번의 게임을 진행
- 1,000번의 게임 후 플레이어의 승률을 계산
- 매 게임 별 플레이어의 소지금 변화를 그래프로 시각화

[4. 작성한 코드]

(과제 1) 1,000번의 게임 진행

generate_episode() 함수를 호출하여 플레이어와 딜러가 어떻게 게임을 진행했는지 그 결과를 resepisode에 저장하였다.

진행 상황을 쭉 출력해보니 아래와 같이 나왔다.

```
[[[(21, True, 1), False, 1]],
 [[(17, True, 2), True, 0], [(17, False, 2), False, 1]],
 [[(17, False, 6), True, 0], [(20, False, 6), False, -1]],
 [[(15, True, 10), True, 0], [(14, False, 10), False, 1]],
 [[(20, False, 9), True, -1]],
 [[(12, False, 2), True, 0],
  [(13, False, 2), True, 0],
  [(20, False, 2), True, -1]],
 [[(12, False, 8), True, -1]],
 [[(21, True, 6), False, 1]],
 [[(15, False, 4), True, 0], [(18, False, 10), False, 1]],
 [[(14, False, 9), True, 0], [(20, False, 9), False, 1]],
 [[(12, True, 10), False, -1]],
 [[(17, False, 8), False, 1]],
 [[(20, True, 8), True, 0], [(19, False, 2), True, -1]],
 [[(15, False, 1), True, -1]],
 [[(17, False, 3), True, 0], [(21, False, 3), True, -1]],
 [[(13, True, 8), True, 0], [(20, True, 10), False, 1]],
 [[(13, False, 10), True, 0], [(18, False, 8), True, -1]],
```

1) reward 값 [0][0][2] = 1 이므로 플레이어의 승으로 첫번째 게임이 끝났다.
2) done값 [1][0][1] = True 임으로 에피소드가 종료되지 않았고 reward 값 [1][0][2] = 0 이므로 에피소드를 계속 진행한다. 두번째 게임도 플레이어의 승으로 끝났다.
3) 세번째 게임은 딜러의 승
4) 네번째 게임은 플레이어의 승
5) 다섯번째 게임은 딜러의 승

1000번의 게임 후 플레이어의 승률을 계산하기 위해 1000번의 반복문 안에서 매 게임이 실행될 때마다 플레이어의 승으로 끝난 게임 수를 카운트 해준다. (위의 예처럼 reward 값이 1인 경우가 플레이어의 승으로 끝난 경우이다.) 코드는 아래와 같다.

```
from pprint import pprint

resepisode = list()
moneyList = list()
# moneyList = []
cnt = 0
playerWincnt = 0
money = 10000  #플레이어 초기 자본은 10,000달러


for i in range(1000):
#     money -= 10    #게임 참가 시 10달러 내야함
    resepisode = mc_es.generate_episode(dealer, agent, deck)

    length = len(resepisode)
    if((length == 1) and (resepisode[0][2]==0 )) :  #무승부로 끝남
        moneyList.append(money)


    else :
        for j in range(length):
            if resepisode[j][2] == 1 :      #플레이어 승
                cnt += 1
                playerWincnt += 1
                money += 10
                moneyList.append(money)
            elif resepisode[j][2] == -1 :    #플레이어 패
                cnt += 1
                money -= 10
                moneyList.append(money)
            else :
                continue
```

generate_episode로 한번의 게임을 실행한다.
length 에는 게임이 몇번만에 끝났는지를 알려준다.

예를 들면, 첫번째 게임은 한번의 턴 만에 플레이어의 승으로 끝이 나서 length = 1 이다.
두번째 게임은 두번의 턴 만에 플레이어의 승으로 끝나므로 length = 2이다.
여섯번째 게임은 세번의 턴 만에 딜러의 승으로 게임이 끝났으므로 length = 3 이다.

게임의 결과로 나올 수 있는 경우의 수는 총 세개가 있다. 즉 무승부, 플레이어가 이기는 경우, 플레이어가 지는 경우이다.

무승부로 끝나는 경우에는 마지막 턴의 reward가 0 일때이고 이 때는 아무것도 해주지 않고 현재 가진 금액을 moneyList에 저장한다. 아무것도 하지 않은 이유는 게임에 참여할 때 10달러를 내고 참여하는데, 무승부로 끝나면 다시 10달러를 획득하여 zerosum 이기 때문이다.

무승부가 아닌 경우에는 reward의 값이 1인지 -1인지에 따라 length 만큼 반복하면서 일을 처리한다.
length 만큼 반복하는 이유는 게임의 턴 마지막에 있는 reward 값을 사용하기 위해서다.

reward = 1 , 즉 플레이어의 승으로 게임이 끝났을 경우에는 승률의 분자, 분모의 수를 +1 씩 해준다. 게임에 참여했을 때 10달러를 내고 게임에 이겨서 20달러를 획득하였으므로 결국에는 10달러를 번 것이고 이를 moneyList에 저장해준다.

reward = -1, 즉 플레이어의 패로 게임이 끝났을 때에는 분모의 수만 +1 해주고 -10달러를 한 금액을 moneyList에 저장한다.

(과제 2) 플레이어의 승률 계산

(과제 2) 플레이어의 승률을 계산

```
############### 코드 작성 ##############################

#플레이어 승률 : (승리수)/(승리수 + 패배수)
win = (playerWincnt / cnt) * 100
print('Win Rate of Player is %f'%win)

#########################################################
```

```
Win Rate of Player is 37.234043
```

과제 조건에서 승률은 승리수/(승리수+패배수) 이라고 했다.
위의 코드에서 계산한 playerWincnt와 cnt로 승률을 계산해주었다.

생각보다 플레이어가 이길 확률이 낮았다.

(과제 3) 플레이어의 소지금 변화를 그래프로 시각화

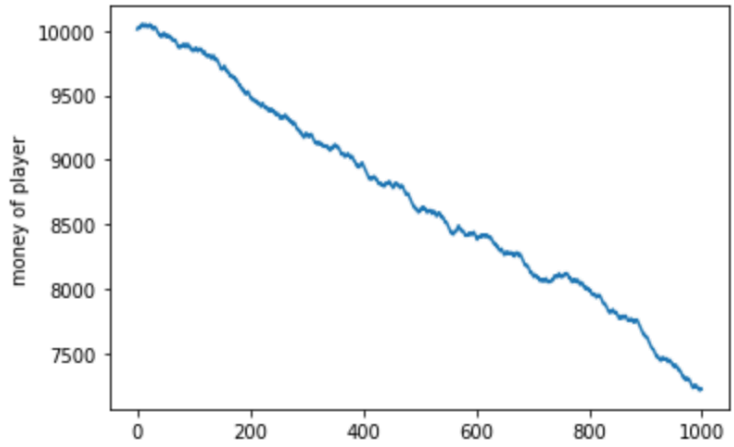
matplotlib.pyplot.plot

• Docs : https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html
• 예제 : https://matplotlib.org/tutorials/introductory/pyplot.html

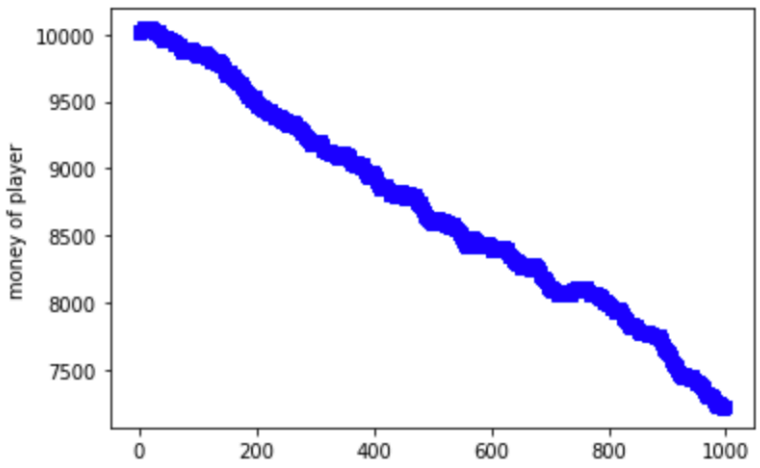
를 참고하여 다양하게 그래프를 작성해보았다.

(과제 3) 플레이어의 소지금 변화를 그래프로 시각화

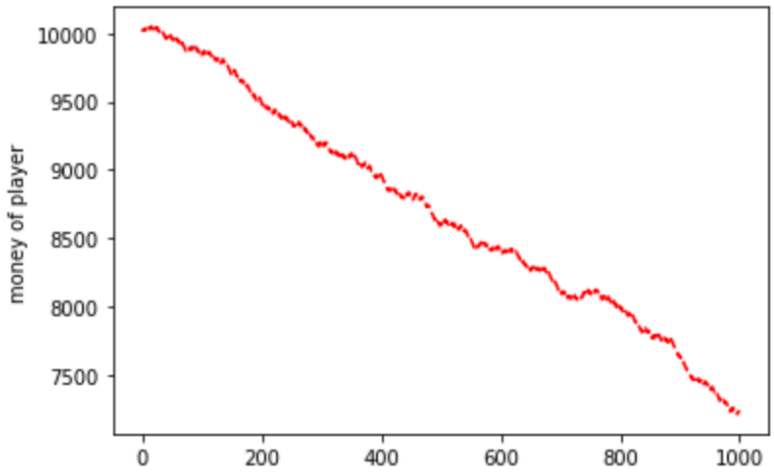
```
############### 코드 작성 ##############################

import matplotlib.pyplot as plt

plt.plot(range(len(moneyList)), moneyList)
plt.ylabel('money of player')
plt.show()

#########################################################
```



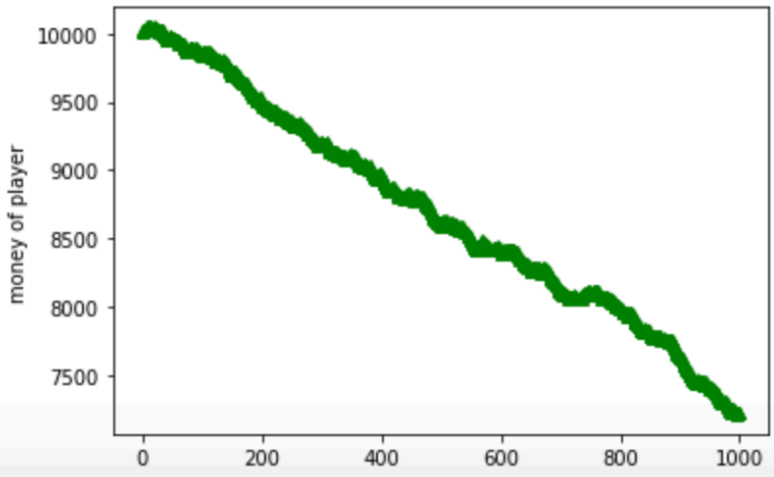

```
plt.plot(range(len(moneyList)), moneyList, 'bs')
plt.ylabel('money of player')
plt.show()
```



```
plt.plot(range(len(moneyList)), moneyList, 'r--')
plt.ylabel('money of player')
plt.show()
```




```
plt.plot(range(len(moneyList)), moneyList, 'g^')
plt.ylabel('money of player')
plt.show()
```



[5. 추가 과제 요구 조건]

: 플레이어의 승률 높이기

State를 수정하여 승률이 더 높은 policy를 찾기
기존 코드의 수정 필요

- 딜러는 게임이 끝났을 때, 남은 카드의 수를 확인
- 15장 이상이라면 해당 덱을 다음 게임에서 그대로 사용
- 15장 미만이라면 52장의 셔플된 새로운 카드를 기존 덱에 추가

```
class MonteCarlo(object):
    def generate_episode(self, dealer: Dealer, agent: Agent, deck: Deck):
        """
        하나의 에피소드(게임)를 생성함
        :param dealer:
        :param agent:
        :param deck:
        :return:
        """

        global k
        k += 1
        #남은 게임 15장 이상이면 덱 그대로 사용, 남은 카드 15장 미만이면 덱 초기화
        remainCard = len(deck.card_deck)

        # 카드 덱, 딜러, Agent를 초기화



        if k==1 :
            deck.reset()
            dealer.reset()
            agent.reset()

        if k>1 and remainCard < 15 :
            deck.reset()
            dealer.reset()
            agent.reset()


        agent.hit(deck)
        agent.hit(deck)
        dealer.hit(deck)
        dealer.hit(deck)

        done = False    # 에피소드의 종료 여부
```

코드를 위와 같이 수정해보았다.
한번의 게임이 실행될 때마다 남은 카드의 수를 검사하여 남은 카드가 15장 미만이 되었을 때 reset을 해준다.
즉 덱에 카드를 초기화 시켜준다.

7. 최적 Policy 학습

```
deck = Deck()
dealer = Dealer()
agent = Agent()
mc_es = MonteCarlo()

mc_es.train(dealer, agent, deck, it=1000000)
```

```
Recent 1000 games win rate :5.061%
 -- 1000 Games WIN : 50 DRAW : 12 LOSS : 938
Total win rate : 4.423%
 -- TOTAL Games WIN : 43800 DRAW : 6792 LOSS : 946408
========== Training : Episode  998000  ===========
Recent 1000 games win rate :4.133%
 -- 1000 Games WIN : 41 DRAW : 8 LOSS : 951
Total win rate : 4.423%
 -- TOTAL Games WIN : 43841 DRAW : 6800 LOSS : 947359
========== Training : Episode  999000  ===========
Recent 1000 games win rate :3.920%
 -- 1000 Games WIN : 39 DRAW : 5 LOSS : 956
Total win rate : 4.423%
 -- TOTAL Games WIN : 43880 DRAW : 6805 LOSS : 948315
========== Training : Episode  1000000  ===========
Recent 1000 games win rate :3.947%
 -- 1000 Games WIN : 39 DRAW : 12 LOSS : 949
Total win rate : 4.422%
 -- TOTAL Games WIN : 43919 DRAW : 6817 LOSS : 949264
```

```
#플레이어 승률 : (승리수)/(승리수 + 패배수)

win = (playerWincnt / cnt) * 100

print('Win Rate of Player is %f'%win)


########################################
```

```
Win Rate of Player is 4.843592
```

그러나 승률이 떨어졌다.

```
remainCard = 52 - len(agent.hands) - len(dealer.hands)
```

그래서 남은 카드의 수를 위와 같이 계산해보았다.

7. 최적 Policy 학습

```
deck = Deck()
dealer = Dealer()
agent = Agent()
mc_es = MonteCarlo()

mc_es.train(dealer, agent, deck, it=1000000)
```

```
Total win rate : 4.458%
 -- TOTAL Games WIN : 44096 DRAW : 6924 LOSS : 944980
========== Training : Episode  997000  ==========
Recent 1000 games win rate :5.522%
 -- 1000 Games WIN : 55 DRAW : 4 LOSS : 941
Total win rate : 4.459%
 -- TOTAL Games WIN : 44151 DRAW : 6928 LOSS : 945921
========== Training : Episode  998000  ==========
Recent 1000 games win rate :4.532%
 -- 1000 Games WIN : 45 DRAW : 7 LOSS : 948
Total win rate : 4.459%
 -- TOTAL Games WIN : 44196 DRAW : 6935 LOSS : 946869
========== Training : Episode  999000  ==========
Recent 1000 games win rate :4.848%
 -- 1000 Games WIN : 48 DRAW : 10 LOSS : 942
Total win rate : 4.460%
 -- TOTAL Games WIN : 44244 DRAW : 6945 LOSS : 947811
========== Training : Episode  1000000  ==========
Recent 1000 games win rate :5.354%
 -- 1000 Games WIN : 53 DRAW : 10 LOSS : 937
```

```
#플레이어 승률 : (승리수)/(승리수 + 패배수)

win = (playerWincnt / cnt) * 100

print('Win Rate of Player is %f'%win)



########################################
```

```
Win Rate of Player is 5.120482
```

그러나 승률이 향상되지 않고 떨어졌다.

```
class Deck(object):
    """
    Deck : Card deck, which can be shuffled, drawn, and reset.
    """


    def __init__(self):
        deck = [2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 11]
        self.card_deck = deck * 4
        self.shuffle()

    def shuffle(self):
        random.shuffle(self.card_deck)

    def draw(self):
        global popCnt
        popCnt += 1
        return self.card_deck.pop()

    def reset(self):
        global popCnt
        if popCnt > 27 :
            deck = [2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 11]
            self.card_deck = deck * 4
            self.shuffle()
            popCnt =0

        self.shuffle()
```

다른 방법을 찾기 위해 Deck 클래스의 reset 함수에 코드를 추가해보았다.
draw 할 때마다 popCnt 수를 센 후, 카드를 27장 넘게, 즉 남은 카드가 15장 미만이 되면 그때 deck 을 초기화 해주었다.

```
#플레이어 승률 : (승리수)/(승리수 + 패배수)

win = (playerWincnt / cnt) * 100

print('Win Rate of Player is %f'%win)



########################################
```

```
Win Rate of Player is 36.984816
```

그러나 승률이 향상되지 않고 초기 상태와 비슷한 것을 확인했다.

Deck 클래스의 reset함수의 마지막 줄 즉, 남은 카드 수가 15장 이상일 때 카드를 셔플해준 부분(self.shuffle())을 지워본 후 다시 코드를 실행해보았다.

```
#플레이어 승률 : (승리수)/(승리수 + 패배수)

win = (playerWincnt / cnt) * 100

print('Win Rate of Player is %f'%win)



#####################################
```

```
Win Rate of Player is 39.079229
```

초기 상태의 37%보다 승률이 약간 높아진 것을 확인할 수 있었다.

[6. 결과 및 고찰]

이번 과제는 블랙잭이라는 카드 게임에 대해 플레이어를 학습시켜서 승률을 계산하는 문제였다.

generate_episode() 함수를 호출하여 플레이어와 딜러가 어떻게 게임을 진행했는지 그 결과를 보고 플레이어의 승률을 계산해야 했는데 generate_episode() 가 리턴하는 값이 어떻게 구성이 되는지 감이 오지 않았다.

그래서 아래와 같이 쭉 출력한 후 시각화 하고 나니 한 눈에 들어와 데이터의 구조를 파악할 수 있었다.

```
[[[(21, True, 1), False, 1]],
 [[(17, True, 2), True, 0], [(17, False, 2), False, 1]],
 [[(17, False, 6), True, 0], [(20, False, 6), False, -1]],
 [[(15, True, 10), True, 0], [(14, False, 10), False, 1]],
 [[(20, False, 9), True, -1]],
 [[(12, False, 2), True, 0],
  [(13, False, 2), True, 0],
  [(20, False, 2), True, -1]],
 [[(12, False, 8), True, -1]],
 [[(21, True, 6), False, 1]],
 [[(15, False, 4), True, 0], [(18, False, 10), False, 1]],
 [[(14, False, 9), True, 0], [(20, False, 9), False, 1]],
 [[(12, True, 10), False, -1]],
 [[(17, False, 8), False, 1]],
 [[(20, True, 8), True, 0], [(19, False, 2), True, -1]],
 [[(15, False, 1), True, -1]],
 [[(17, False, 3), True, 0], [(21, False, 3), True, -1]],
 [[(13, True, 8), True, 0], [(20, True, 10), False, 1]],
 [[(13, False, 10), True, 0], [(18, False, 8), True, -1]],
```

초기 상태에서 플레이어의 승률은 약 37%가 나왔는데 생각보다 승률이 낮았다. 그래서 그래프로 소지금을 시각화하여 표현했을 때에도 상승하는 곳 없이 꾸준히 소지금이 감소하는 것이 보였다.

추가과제의 목적인 더 높은 policy를 찾기 위해 남은 카드의 수에 따라서 덱을 초기화할지 그대로 쓸지 정해야했다. 남은 카드의 수를 계산하기 위해 다양하게 시도한 결과 처음의 37%보다 약간 상승한 39%의 승률이 결과로 나왔다. 2% 향상이 된 것이 추가과제의 목적에 부합하는지는 잘 모르겠다.

처음에 MonteCarlo 클래스에서 남은 카드의 수를 계산하는 방법으로 승률 향상을 시도하였을 때, 승률이 한 자리수로 떨어졌었다. 아직 이 원인이 무엇인지 파악을 하지 못해서 아쉽다.

이 게임을 한 번도 해보지 않아서 생소했지만 과제를 통해 유명한 카드 게임에 머신러닝으로 모델을 학습시키는 의미있는 경험을 할 수 있어서 좋았다. 알파고처럼 사람과 대결해서 이길 수 있는 블랙잭 머신이 나오길 기대한다.