**REPORT**

**| 데이터마이닝**

**| Deadline : 2020/06/22**

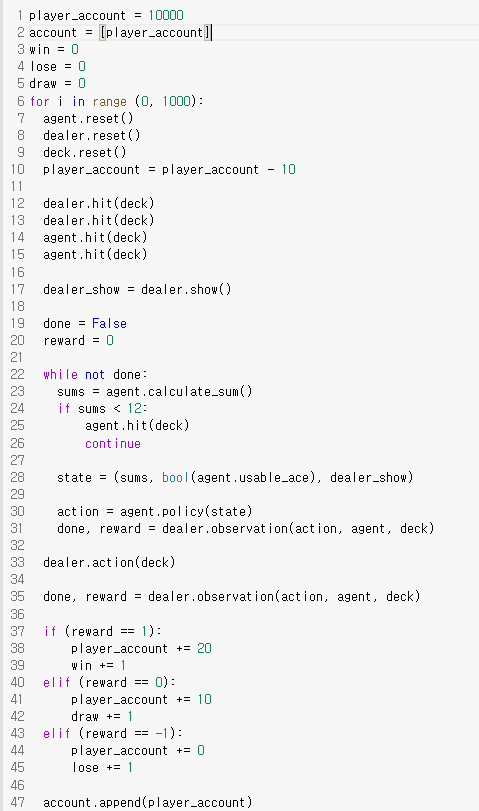
**| Black Jack**

**| 12171598 김용준**

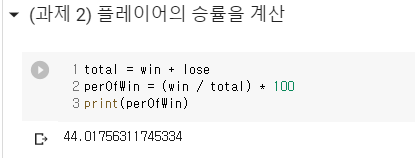
**| 과제(필수)**

# [코드 설명] :

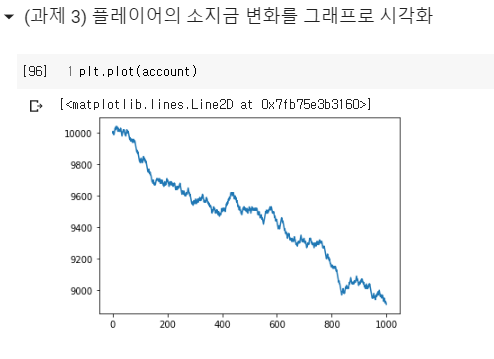




1번 과제의 코드입니다. 학습시킨 policy를 바탕으로 1000번의 게임을 수행하였고 agent카드의 합, ace 여부, dealer가 보여준 카드 세가지의 state를 통해 최적의 action을 얻어 게임을 진행했습니다.



플레이어의 승률을 계산한 결과 대략 44퍼센트정도의 승률을 얻었습니다.



게임진행에 따른 소지금의 변화는 위의 그래프와 같았습니다.

# [실험 결과 및 고찰] :

이번 과제에서 Monte-Carlo Control을 통해 모든 state, action pair로부터 최적의 policy를 구한 후 블랙잭 게임을 수행했습니다. 이후 agent의 승률을 계산하고 배팅액이 일정할 때 게임에 따른 소지금의 변화를 시각화 하였습니다. 세가지의 state를 가지고 40퍼센트 정도의 승률을 얻었는데 더 다양한 state를 이용한다면 더욱 좋은 성적을 낼 수 있을 것 같다고 생각하였고 카드 카운팅을 적용해 보았습니다.

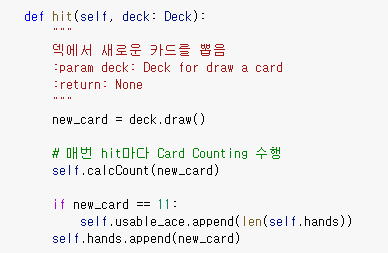
**| 과제(Optional)**

# [수정한 State 및 코드 설명] :

기존 세가지 state에 추가로 카드카운팅 넘버를 state에 추가하여 네가지의 state로 최적의 policy를 구했습니다.

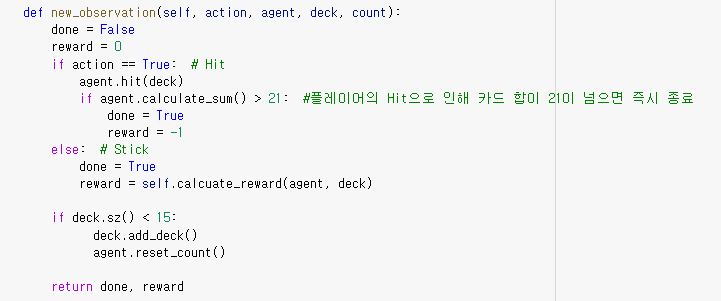


위와 같이 card\_counting array를 만들고 현재 agent의 카드 카운팅 값을 리턴하는 getCount함수와 카드를 hit 했을 때 카드 카운팅 값을 추가하는 calcCount함수를 추가했습니다.

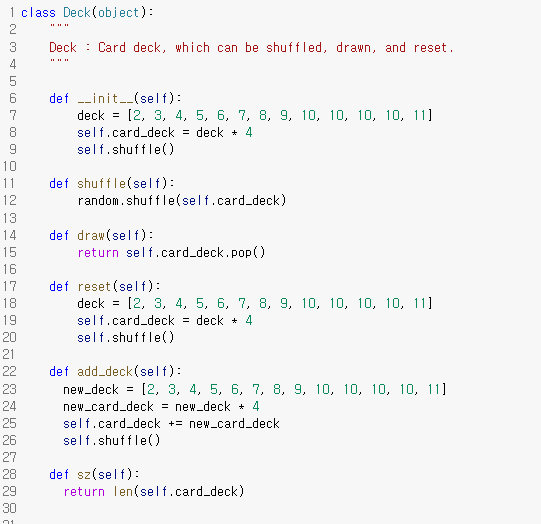


이후 agent가 hit할 때마다 calcCount를 실행해주었습니다.

또한 Dealer클래스에서 observation함수를 다음과 같이 수정하였습니다.

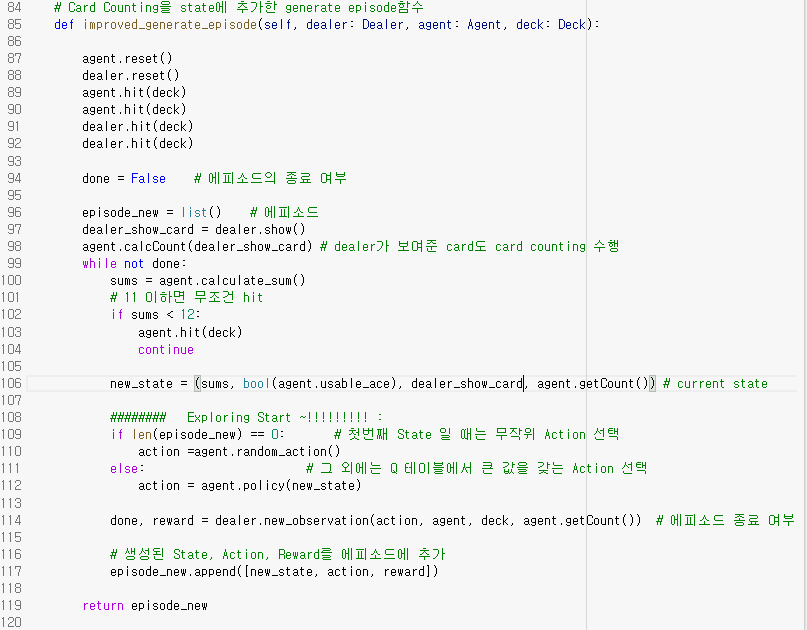


위의 new\_observation은 기존 observation에서 덱의 사이즈에 따라 덱을 추가할지 그대로 유지할지 결정하는 코드를 추가하였고 add\_deck함수는 아래와 같습니다. agent.reset\_count함수는 agent의 카드카운팅 값을 초기화하는 함수입니다.



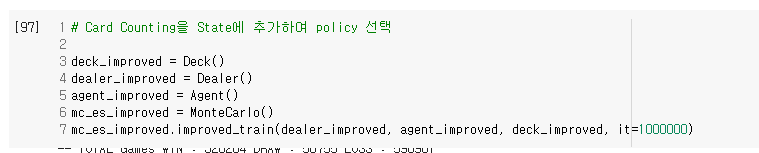
Deck클래스에서 add\_deck함수를 위와 같이 생성했습니다.

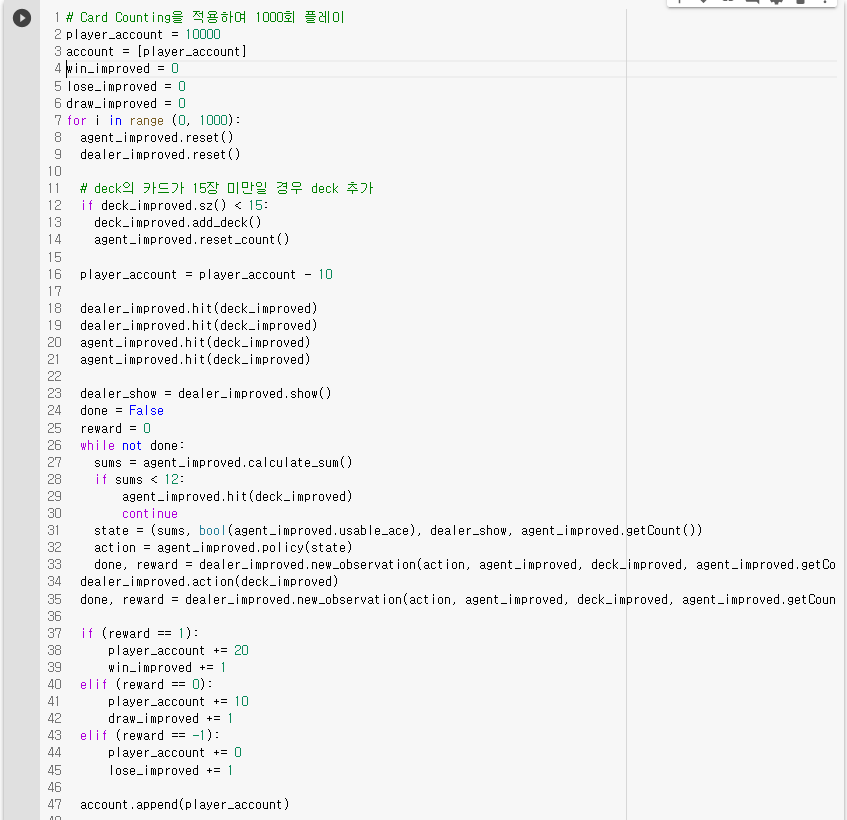
또한 train함수내 generate\_episode함수를 아래와 같이 수정하였습니다.



New\_state를 sums, usable\_ace, dealer.show, agent.getCount로 수정하여 네가지 state를 통해 policy를 구하도록 하였습니다. Done, reward의 결정은 위에서 설명한 new\_observation함수를 통해 결정하였습니다.

이후 기존 과제와 같이 policy를 학습시킨후 1000번의 게임을 통해 승률을 계산하고 이를 시각화 하였습니다.







카드 카운팅을 통해 승률이 4퍼센트 상승했으며 최종 소지금역시 증가하였습니다.

# [실험 결과 및 고찰] :

State를 추가 한 후 유의미한 승률 상승이 있었지만 기대하였던 만큼 획기적인 변화가 있지는 않았습니다. 하지만 이번 과제를 통해 Monte-Carlo Control에서 state의 설정이 결과에 어느정도 영향을 미칠 수 있는지 경험하였고 직접 알고리즘을 설계함으로써 Monte-Carlo에 대해 확실하게 이해할 수 있었습니다.