**강화학습**

1. [강화학습의 정의](#강화학습의정의)

2. [강화 학습에서 사용하는 용어](#강화학습에서사용하는용어)

3. [강화 학습에 필요한 요소](#강화학습에필요한요소)

4. [강화 학습을 구현한 게임 1 - Tic Tac Toe](#강화학습을활용할게임1)

5. [강화 학습을 구현한 게임 2 - Ping Pong](#강화학습을구현한게임2)

6. DQN으로 구현한 게임 1 - Invader

7. DQN으로 구현한 게임 2 - 벽돌깨기

**1. 강화학습의 정의**

(1) 행동 심리학에서의 강화학습(Behavioral Psychology) : 강화는 동물이 시행착오를 통해 학습하는 방법 중 하나로 강화라는 개념을 처음 제시한 것은 스키너라는 행동 심리학자이다. 강화라는 것은 동물이 이전에 배우지 않았지만 직접 시도하면서 행동과 그 결과로 나타나는 좋은 보상 사이에 상관관계를 학습하는 것이다. **강화의 핵심은 바로 보상을 얻게 해주는 행동의 빈도의 증가이다**. 다른 말로 이야기 하면 보상을 얻게하는 행동을 점점 더 많이 하도록 학습하는 것을 말한다.

(2) 인공지능에서의 강화학습(Artificial Intelligence)

1) 인공지능의 정의

지능 : 문제를 해결할수 있는 능력

인공지능 : 지능 작업을 수행할 수 있는 기계의 능력

2) 인공지능의 구현 방법(합리주의 vs 경험주의)

① 지식공학방식(Top-down방식) : 특정 분야의 장인들이 학문연구나 오랜 실무 경험으로 터득한 지식을 직접 컴퓨터에 넣어주어야 컴퓨터가 인종지능 역할을 할 수 있다는 원리(합리주의자 주장)

② 머신러닝방식(Bottom-up방식) : **컴퓨터가 일일이 코드로 명시하지 않은 동작을 데이터로부터 학습해서 실행할 수 있다**는 원리(경험주의자 주장)

3) 머신러닝의 종류

① 지도학습 : 사람이 레이블을 알려주고 데이터를 분류하라고 학습시키는 것

② 비지도학습 : 사람이 레이블을 주지않고 컴퓨터가 스스로 데이터의 패턴을 발견하여 데이터를 분류하는 것

③ 강화학습 : 시행착오를 통해서 환경에 적응하는 학습방법, 지도학습은 이 행동이 맞다라는 직접적인 정답을 통해 오차를 계산해서 학습했지만 강화학습은 자신의 행동의 결과로 나타나는 보상을 통해서 행동의 바람직함을 스스로 알아가는 학습 방법이다.

(3) 제어이론에서의 강화학습(Control Theory) : 제어이론이란 동적 시스템의 거동을 다루는 이론으로 주어진 시스템에 대하여 그 입력을 조절함으로써 그 출력을 원하는 대로 조절하는 제어기(Controller)를 만드는 데에 적용된다. 시스템의 실제 출력값을 원하는 값과 비교하여 그 차이를 시스템의 입력값 조절에 반영하는 방식의 제어기를 폐루프 제어(Closed-loop control)라고 한다. 예를 들어 자동차의 속도를 측정하여, 그 속도가 원하는 속도보다 낮으면 속도를 더 내게 하고, 원하는 속도보다 높으면 속도를 덜 내게 하는 방식의 제어를 하는 것이다.

드론의 제어에 사용되는 PID 계수를 자동으로 맞춰주는 방법이 없을까?에 관련한 문제도 강화학습으로 해결 가능

(4) 운용과학에서의 강화학습(Operations Research) : 운용 과학이란 수학적, 통계적 모형 등을 활용하여 효율적인 의사결정을 돕는 기법이다. OR의 대상 분야 중에 하나로써 동적계획법이 있는데 원리는 매우 간단하다. 일반적으로 주어진 문제를 풀기 위해서, 문제를 여러 개의 하위 문제로 나누어 푼 다음, 그것을 결합하여 최종적인 목적에 도달하는 것이다.

(5) 신경과학에서의 강화학습(Neuroscience) : 신경과학은 뇌를 포함한 모든 신경계에 대해서 연구하는 학문이다. 신경과학은 우리가 외부 환경을 어떻게 인지하며 경험하는지, 그리고 다른 사람과 어떻게 상호관계를 맺는지 등을 밝혀내는 데 중점을 두고 있다.

**2. 강화학습에서 사용하는 용어**

(1) 에이전트(agent) : 강화학습을 통해 스스로 학습하는 컴퓨터 (학습하는 주체)

(2) 환경(environment) : 에이전트가 행동하는 곳

(3) 상태(state) : 에이전트가 처한 특정한 상황에 대한 관찰

ex) 핑퐁에서 공이 왼쪽에서 오른쪽으로 2의 속도로 오고있다.

(4) 행동(action) : 에이전트가 어떠한 상태에서 취할 수 있는 행동

ex) 핑퐁에서 공이 왼쪽에서 오른쪽으로 갈 때 패들이 어느 쪽으로 갈건지 정하는 액션

(5) 보상(reward) : 행동에 대한 보상, 강화학습을 다른 머신러닝 기법과는 다르게 만들어주는 가장 핵심적인 요소

1) 즉각적 보상 2) 최종 보상

(6)정책(policy) : agent는 어떤 state에 도착하면 action을 결정하는 데 어떤 state에서 어떤 action을 할 지를 결정하는 것을 policy라고 한다. 강화학습의 목적은 최적의(reward를 최대화하는) 정책을 찾는 것이다.

즉 강화 학습(Reinforcement learning)은 이 다루는 문제 중에서 다음과 같이 기술되는 것을 다룬다. 어떤 환경을 탐색하는 에이전트가 현재의 상태를 인식하여 어떤 행동을 취한다. 그러면 그 에이전트는 환경으로부터 보상을 얻게 된다. 보상은 양수와 음수 둘 다 가능하다. 강화 학습의 알고리즘은 그 에이전트가 앞으로 누적될 보상을 최대화 하는 일련의 행동으로 정의되는 정책을 찾는 방법이다.

**3.** **강화학습의 특징**

(1) 시행착오(Trial and Error) : agent가 해보지 않고 예측하고 action을 하는 것이 아니라 action을 일단 해보면서 자신을 조정해나가는 것이다. 만약 좋은 action을 취했을 경우에는 좋은 반응이 environment 로부터 오게 되는데 이 반응을 reward 라고 한다. 사람이 그러하듯이 인공지능 또한 상을 많이 받기 위해 노력할 것이다. 어떻게 하면 상을 더 많이 받을 것이냐? 는 강화학습의 핵심 쟁점중의 하나이다.

(2) 지연된 보상(Delayed Reward) : 두 번째는 강화학습이 다루는 문제에 "시간"이라는 개념이 포함되어 있다는 것과 관련이 있다. 강화학습은 시간의 순서가 있는 문제를 풀기 때문에 지금 한 행동으로 인한 환경의 반응이 늦어질 수가 있는데(혹은 다른 행동과 합해져서 더 좋은 환경의 반응을 받아낼 수도 있습니다), 이럴 경우에 환경이 반응할 때까지 여러 가지 다른 행동들을 시간의 순서대로 했기 때문에 어떤 행동이 좋은 행동이었는지 판단하기 어려운 점이 있습니다. 이 점은 강화학습의 중요한 문제로서 계속 머리속에 넣어둘 필요가 있습니다.

**4. 강화학습의 대상**

(1) 적용 대상 : 강화 학습은 마치 사람처럼 환경과 상호작용하면서 스스로 학습하는 방식인데 이러한 학습은 과연 어떤 문제에 적용해야 하는가?

-> 강화 학습은 결정을 순차적으로 내려야 하는 문제에 적용된다.

(2) 순차적 행동 결정 문제 : 현재 위치에서 행동을 한번 선택하는 것이 아니라 계속적으로 선택해서 최종적으로 골인지점에 도달하는 것을 결정하는 문제를 말한다.

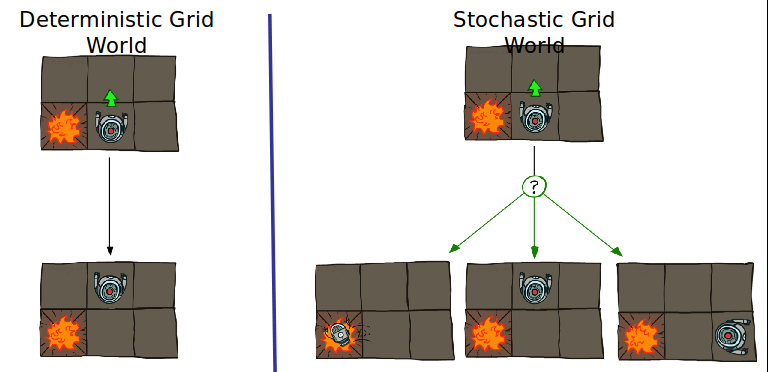
-> 이 문제를 해결하는 방법에는 다이나믹 프로그래밍, 진화 알고리즘, 강화 학습이 있는데 앞의 두가지 방법에는 각기 한계를 가지고 있으며 강화학습이 그 한계를 극복하는 문제 해결 방안으로써 최근에 각광받고 있다.

(3) MDP(MarKov Decision Process) : 순차적 행동 결정문제를 해결하기 위해 (에이전트가 학습하고 발전하기 위해) 이 문제를 수학적으로 표현한 것으로써 5가지 요소를 가지고 있다.

1) 상태(State) : state는 agent가 인식하는 자신의 상태이다. 사람으로 치면 눈이라는 관측도구를 통해서 '나는 방에 있어' 라고 인식하는 과정에서 방이 state가 된다. 이외에도 state 는 생각보다 많은 것들이 될 수 있는데 달리는 차 같은 경우에는 '차는 Jeep이고 사람은 4명 탔으며 현재 100km/h로 달리고 있다' 라는 것이 state 가 될수 있다. atari game 같은 경우에는 게임화면 자체, 즉 pixel agent 가 인식하는 state 가 된다. 즉 문제는 정의하기 나름이다.

2) 행동(Action) : agent의 역할은 environment 에서 특정 state에 갔을 때 action 을 지시하는 것으로써 robot이 왼쪽으로 갈지, 오른쪽으로 갈지를 결정하는 것과 같다. 이때 오른쪽으로 간다 or 왼쪽으로 간다 라는 것이 action이 되고 agent 가 그 action 을 취했을 경우에 실제로 오른쪽이나 왼쪽으로 움직이게 된다.

3) 상태변환확률(State transition probability matrix) : robot이 action을 취하면 environment상의 agent의 state가 변하는데 그것 또한 environment가 agent에게 알려준다. 정확히 말하면 agent가 observe 하는 것이다. 대신에 어떠한 외부요인에 의해(ex : 바람이 분다던지 등) robot이 왼쪽으로 가려했지만 오른쪽으로 가는 경우가 발생할 수 있다. 아래의 그림을 참고해보자. 로봇은 앞으로 간다고 갔지만 왼쪽으로 가서 불에 빠질 수도 있고 오른쪽으로 갈 수도 있다는 것이다. 이러한 확률을 표현하는 것이 state transition probability matrix이다.



4) 보상함수(Reward) : agent 가 action을 취하면 그에 따른 reward를 environment가 agent에게 알려준다. 그 reward는 atari game에서는 score, 바둑의 경우는 승패(알파고가 학습하는 방법), trajectory control의 경우에는 의도한 궤도에 얼마나 가깝게 움직였는지가 된다. 보상함수의 수식은 아래와 같다. s라는 state에 있을 때 a는 action을 취했을 때 얻을 수 있는 reward이다.



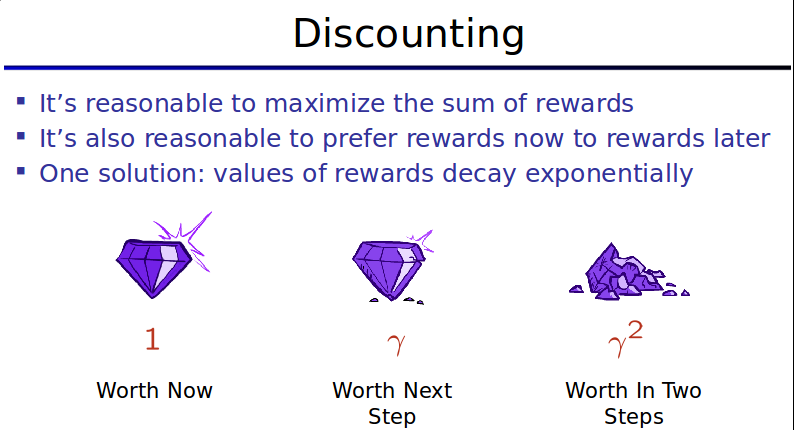
이 reward를 imediate reward라고 하는데 agent는 단순히 즉각적으로 나오는 reward만 보는 것이 아니라 이후로 얻는 reward들까지 고려한다.

5) 감가율(Discount Factor) : reward의 정의에 따라 각 state에서 어떠한 action을 취하면 reward를 받게 되는데 이때 단순히 받았던 reward들을 더하면 다음과 같은 문제가 발생한다.

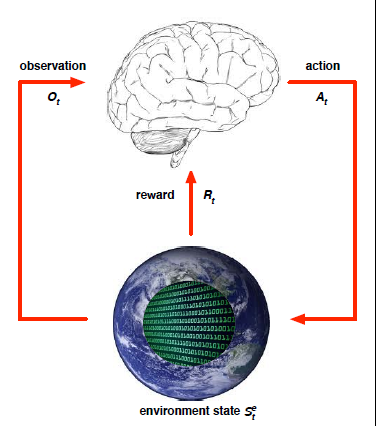
- 어떠한 agent는 각 time-step마다 0.1씩 reward를 받고 다른 agent는 1씩 받았을 경우에 시간이 무한대로 흘러간다면 0.1씩 계속 더해도 무한대이고 1씩 계속 더해도 무한대이다. 수학에서 무한대는 크기를 비교 할 수 없 다.

- agent가 episode를 시작하자마자 1 받았을 경우와 끝날 때 1을 받았을 경우를 둘 다 전체 reward를 1을 받았기 때문에 두 상황 중에 어떤 경우가 더 나은 건지를 판단할 수 없다.

이러한 두가지 경우 때문에 discount factor 라는 개념이 등장한다. 사람의 입장에서 생각해보면 당장 지금 배고픈 것을 채우는 것이 내일 배고픈 것을 채우는 것보다 중요하다 생각하고 행동하는 것처럼 discount factor는 0에서 1사이의 값 입니다. 아래의 그림을 보면 이해가 쉽다.



discount factor가 0이면 상당히 근시안적인 것이고 discount factor가 1이면 상당히 미래지향적인 것이라서 사실은 사람이 어떤 행동을 결정할 때 미래를 생각하며 결정하긴 하지만 모든 미래에 일어날 일을 다 고려하지는 않습니다. 따라서 discount factor는 보통 0에서 1사이의 값을 사용합니다. 이렇듯 agent는 action을 취하고 state를 옮기고 reward를 받고 하면서 environment 와 상호작용을 하는데 그 그림은 아래와 같다.



agent가 observation을 통해서 자신의 state를 알게 되면 그 state에 맞는 action을 취하게 된다. 학습을 하지 않은 초기에는 random action을 취한다. 그러면 environment가 agent에게 reward와 다음 state를 알려주게 된다.

6) 정책(Policy) : agent는 어떤 state에 도착하면 action을 결정하는데 어떤 state에서 어떤 action을 할 지를 policy라고 한다. 결국에 강화학습의 목적은 optimal policy(accumulative reward = return 을 최대화하는 policy)를 찾는 것이다.

**4. 강화 학습을 구현한 게임 1 - Tic Tac Toe**

(1) [TIC TAC TOE 기본 코드의 이해](#tictactoe기본코드)

(2) [일반학습을 적용한 TIC TAC TOE 코드](#일반학습을적용한코드)

(3) [강화학습을 적용한 TIC TAC TOE 코드](#강화학습을적용한코드)

(1) TIC TAC TOE 기본 코드의 이해

※ TIC TAC TOE 프로그램을 만들기 위해서 필요한 기능

1) 보드판 그리는 기능 : [printboard 함수](#printboard함수)

2) 보드판을 리셋하는 기능 : [emptystate 함수](#emptystate함수)

3) 중간중간 게임의 승리자가 결정되었는지 확인하는 기능 : [gameover 함수](#gameover함수)

4) 어디에 수를 둘지 물어보고 보드판에 두게 해주는 기능 : [action 함수](#action함수)

5) 게임이 다 끝나면 누가 이겼는지 알려주는 메세지를 출력하는 기능 : [episode\_over 함수](#episode_over함수)

6) 게임 순서에 맞춰서 x 다음에 o 가 수행되고 play 하는 기능 : [play 함수](#play함수)

7) 컴퓨터가 임의로 수를 두도록 하는 기능 : [random 함수](#random함수)

8) 게임을 진행하게 하는 기능 : [human, computer class 의 객체화](#class객체화)

9) 게임이 한번 만에 끝나는게 아니라 계속 진행이 되게 하는 기능 : [무한루프문](#무한루프문)

10) 기보 내용을 저장하게 하는 기능 : [CSV writing](#csvfilewriting)

1) **printboard 함수** : 보드판을 출력하는 함수

EMPTY = 0

PLAYER\_X = 1

PLAYER\_O = 2

DRAW = 3

BOARD\_FORMAT = "----------------------------\n"\

"| {0} | {1} | {2} |\n"\

"|--------------------------|\n"\

"| {3} | {4} | {5} |\n"\

"|--------------------------|\n"\

"| {6} | {7} | {8} |\n"\

"----------------------------"

NAMES = [' ', 'X', 'O']

# 보드 출력

def printboard(state):

cells = []

for i in range(3):

for j in range(3):

cells.append(NAMES[state[i][j]].center(6))

# cells 에 names 의 값을 6칸 중 center 위치에 채워 넣고 있다.

#print(cells)



#print(\*cells)



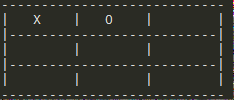
# cells 리스트 안의 요소 전부 가져온다.(빈칸도 전부 가져온다)

print(BOARD\_FORMAT.format(\*cells))

# BOARD\_FORMAT 에 cells 요소를 차례차례 대입시켜준다.

state = [[1,2,0],[0,0,0],[0,0,0]]

print(printboard(state))



2) **emptystate 함수** : 비어있는 판을 출력하는 함수 (게임이 끝났으면 리셋함)

EMPTY = 0

def emptystate():

return [[EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY]]

결과 :



- 이 함수를 활용하여 비어있는 9칸의 게임판 만들기

EMPTY = 0

def emptystate():

return [[EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY]]

state = emptystate()

def printboard(state):

cells = []

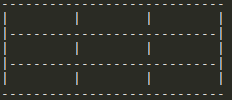
for i in range(3):

for j in range(3):

cells.append(NAMES[state[i][j]].center(6))

print(BOARD\_FORMAT.format(\*cells))

print(printboard(state))



3) **gameover 함수** : 누가 이겼는지를 결정하는 함수

def gameover(state):

# 가로/세로로 한 줄 완성한 플레이어가 있다면 그 플레이어 리턴

for i in range(3): # 이 for 문에서는 가로,세로 3줄을 체크해주고 있다.

if state[i][0] != EMPTY and state[i][0] == state[i][1] and state[i][0] == state[i][2]: # 가로 3줄 확인

return state[i][0]

if state[0][i] != EMPTY and state[0][i] == state[1][i] and state[0][i] == state[2][i]: # 세로 3줄 확인

return state[0][i]

# 좌우 대각선 확인

if state[0][0] != EMPTY and state[0][0] == state[1][1] and state[0][0] == state[2][2]:

return state[0][0]

if state[0][2] != EMPTY and state[0][2] == state[1][1] and state[0][2] == state[2][0]:

return state[0][2]

# 판이 비었는지 (승패가 결정이 나지 않은 게임 진행의 상황)

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

return EMPTY

return DRAW # 판은 가득 차있는데 승패가 나지 않은경우

- gameover 함수에 state 값을 넣고 출력되는 결과를 테스트 해보기

In 
In 
In 
In 
In 
In 
[66] : 
[69] : 
[91] : 
[92] : 
[93] : 
state 
print (gameover(state) ) 
state 
print (gameover(state) ) 
state 
print (gameover(state) ) 

4) **action 함수** : 어디에 수를 둘지를 물어보는 함수(human class 내)

def action(self, state):

printboard(state)

action = None

while action not in range(1, 10):

# 액션의 범위가 1~10이 아닌경우에 돌려라

action = int(input('Your move? '))

switch\_map = { 1: (0, 0), 2: (0, 1), 3: (0, 2),

4: (1, 0), 5: (1, 1), 6: (1, 2),

7: (2, 0), 8: (2, 1), 9: (2, 2) }

return switch\_map[action]

]

- 내 턴에 아무것도 입력하지 않으면 계속 물어보게 하기

def action(self, state):

printboard(state)

action = None

while action not in range(1, 10):

try :

action = int(input('Your move? '))

except ValueError :

continue

switch\_map = {1: (0, 0), 2: (0, 1), 3: (0, 2),

4: (1, 0), 5: (1, 1), 6: (1, 2),

7: (2, 0), 8: (2, 1), 9: (2, 2) }

return switch\_map[action]

- 이미 두어진 자리에 수를 두면 '이미 둔수입니다.' 라는 메세지가 뜨고 다시 물어보게 하기

def action(self, state):

printboard(state)

action = None

switch\_map = {1: (0, 0), 2: (0, 1), 3: (0, 2),

4: (1, 0), 5: (1, 1), 6: (1, 2),

7: (2, 0), 8: (2, 1), 9: (2, 2) }

cycle = True

while action not in range(1, 10) or cycle:

try :

action = int(input('Your move? '))

if state[switch\_map[action][0]][switch\_map[action][1]] != 0 :

print('이미 둔 수입니다.')

else :

cycle = False

except ValueError :

print('허용값이 아닙니다. 다시 입력하세요.')

continue

except KeyError :

print('허용범위를 초과하였습니다. 다시 입력하세요.')

continue

return switch\_map[action]

5) **episode\_over 함수** : 게임 종료시 승패 여부 메세지를 출력하는 함수

winner 변수에 들어오는 값이 1 ,2 draw 이다.

winner = play(p1, p2)

def episode\_over(self, winner):

if winner == DRAW:

print('Game over! It was a draw.')

else:

print('Game over! Winner: Player {0}'.format(winner))

6) **play 함수** : 게임을 진행하는 함수 (o 다음에 x 다 라는 순서를 지정),

실제로 게임을 진행하는 함수인데 게임 종료 여부를 확인하고 결과를 리턴함.

def play(agent1, agent2):

human class computer class

state = emptystate() # 판을 새로 리셋

for i in range(9): # 0,1,2,3,4,5,6,7,8

if i % 2 == 0: # i 가 짝수이면 수행

move = agent1.action(state)

# 인스턴스화한 human class의 액션을 수행하겠다.

# (0,1) 의 값을 move 에 입력한다.

else: # i 가 홀수이면 수행

move = agent2.action(state)

# 인스턴스화한 human class의 액션을 수행하겠다.

# (0,1) 의 값을 move 에 입력한다.

state[move[0]][move[1]] = (i % 2) + 1

#state[0][1] = (i % 2)+1

#state = [[0,0,0],[0,0,0],[0,0,0]]

#state[0][1] = (0%2) + 1 = 1 , i 가 짝수일 때

#state = [[0,1,0],[0,0,0],[0,0,0]]

#state[0][1] = (1 % 2)+1 = 2 , i 가 홀수일 때

#state = [[0,2,0],[0,0,0],[0,0,0]]

winner = gameover(state)

# play 중간중간마다 게임 종료 여부를 확인해주고 있다.

if winner != EMPTY:

return winner

return winner

7) **random 함수** : 컴퓨터가 임의로 수를 두게 하는 함수 (computer class내)

# state = [[1,0,0],[0,0,0],[0,0,0]] 플레이어가 착점한 수

def random(self, state):

available = [] # 비어있는 리스트 변수 선언

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

available.append((i, j))

# [(0,1),(0,2),(1,0),..] 이렇게 09 개 만들어서 비어있는 리스트에 채워 넣음

return random.choice(available)

# 비어있는 것 중에 하나를 랜덤으로 고른다.

# (0,0) (0,1) (0,2) 3개가 비어있으면 이중에 하나를 임의로 선택한다.

# state = [[1,0,2],[0,0,0],[0,0,0]] , 임의로 (0,2) 를 컴퓨터가 선택한 결과

8)**class 객체화** : Tic Tac Toe 게임을 진행하게 하는 기능

- 휴먼 클래스와 컴퓨터 클래스를 객체화해서 게임을 진행 시키는 방법

if \_\_name\_\_ == "\_\_main\_\_": # 내가 실행하는 모듈이 main 모듈일 때 실행하라.

p1 = Human(1) # 내가 실행하고 있는 창이 main 모듈이다.

p2 = Computer(2)

# 위의 스크립트들을 tictactoe.py 로 저장해서 다른 실행창에서 import tictactoe 로 수행했을 때에는 if \_\_name\_\_ == "\_\_main\_\_": 이하 아래의 코드는 실행하지 말아라 라는 코드

 - 컴퓨터끼리 게임을 진행 시키는 방법

if \_\_name\_\_ == "\_\_main\_\_":

p1 = Computer(1)

p2 = Computer(2)

9) **무한loop문** : 게임이 계속 지속되도록 하는 기능

while True: #무한루프를 돌리며 계속 플레이를 시키고 있다.

winner = play(p1, p2) # winner 에 게임결과가 들어오고 있다.

episode\_over(winner) # 결과에 대한 메세지를 episode\_over 에서 출력

10) **CSV file writing** : 기보 내용을 저장하게 하는 기능

- 게임 과정의 데이터(틱텍토 기보)를 d://data//test01.csv 라는 이름으로 저장

# 컴퓨터 클래스의 액션부분에 아래의 코드를 추가함

class Computer(object):

def \_\_init\_\_(self, player):

self.player = player

def random(self, state):

available = [] # 비어있는 리스트 변수 선언

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

available.append((i, j))

return random.choice(available)

def action(self, state):

printboard(state)

move = self.random(state)

state[move[0]][move[1]] = self.player

**f = ("d://data//test01.csv")**

**w = csv.writer(open(f,'a'),delimiter=',')**

**w.writerow([state[0][0],state[0][1],state[0][2],**

**state[1][0],state[1][1],state[1][2],**

**state[2][0],state[2][1],state[2][2]])**

return move

결과 : 100 000 000

120 000 000

120 010 000

:

122 010 001  --> 이런식으로 게임의 전 과정이 csv 파일로 기록됨

- 게임의 진행과정 말고 게임의 결과만 test02.csv 라는 이름으로 저장하는 법

(게임결과 승,패,무도 같이 출력되도록)

# play 함수부분에 아래의 코드를 추가함

import csv

def play(agent1, agent2):

state = emptystate()

for i in range(9):

if i % 2 == 0:

move = agent1.action(state)

else:

move = agent2.action(state)

state[move[0]][move[1]] = (i % 2) + 1

winner = gameover(state)

if winner != EMPTY:

**f = ("d://data//test02.csv")**

**w = csv.writer(open(f,'a'),delimiter=',')**

**w.writerow([state[0][0],state[0][1],state[0][2],**

**state[1][0],state[1][1],state[1][2],**

**state[2][0],state[2][1],state[2][2],winner])**

return winner

return winner

(2) 일반학습을 적용한 TIC TAC TOE 코드

※ 일반 학습을 적용하기 위해 알아야 할 알고리즘

1) **min/max 알고리즘**

① 턴제 게임에서 사용할 수 있는 알고리즘으로써 만들 수 있는 모든 경우의 수를 찾아본 후 상대가 최고의 수를 둔다고 가정하고 가장 좋은 수를 찾아 나가는 알고리즘이다. 좀더 자세히 살펴보자면 (플레이어가 이길수 있는 확률) - (컴퓨터가 이길 수 있는 확률)을 계산하여 모든 확률에서 플레이어는 가장 큰 값을 선택하고 컴퓨터는 가장 낮은 값을 선택 하는 것 이다.

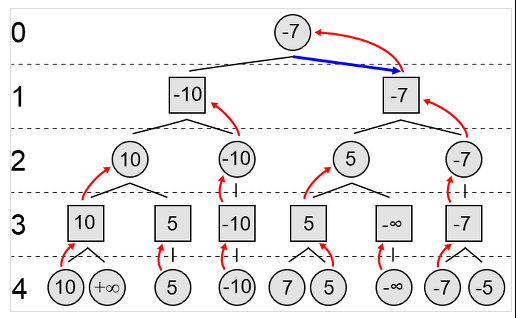
② 상태를 p라고 가정했을 때, 상태 함수 e(p)는 다음과 같다.

e(p) = max(플레이어의 턴에 플레이어가 이길 수 있는 수) - min(컴퓨터 턴에 플레이어가 이길 수 있는 수)

e(p) = +∞ (플레이어가 이기는 상황)

e(p) = -∞ (컴퓨터가 이기는 상황)

③ min/max알고리즘을 그림으로 살펴보면 다음과 같다.



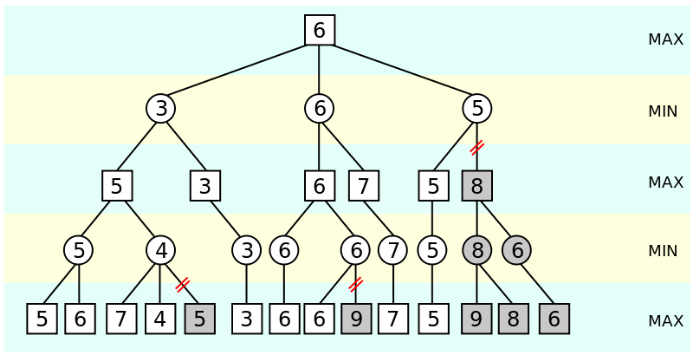
ⓐ 초기 위치에서 위의 트리같이 가능한 경우를 모두 나열한다. 경우 밑에 또 가능한 경우들을 나열한다. 이 그림 예시에서는 4단계까지 나열했다.

ⓑ 각 경우의 수마다 점수를 매긴다. 예를들어 오목이라면, 「내 돌이 3개 연속해있으면 +2점, 내 돌이 4개 연속 +5점, 내 돌이 5개 연속이면 +20점, 상대 돌이 3개 연속해있으면 -2점, 상대 4개 연속 -5점, 상대 5개  연속이면 -20점」 이런 식으로 점수를 매긴다

ⓒ 플레이어가 min/max를 쓴다고 하면, 가장 밑에서부터 각 노드마다 플레이어의 턴에는max의 값을 가져오고, 컴퓨터의 턴에는 min의 값을 가져온다. 아래와 같이 max와 min의 값을 번갈아가면서 가져오고, 마지막은 플레이어의 턴이므로 max의 값을 갖는 경우를 선택한다.

④ alpha/beta pruning : tictactoe는 게임의 경우의 수가 많지 않아 전부 min/max 알고리즘을 쓰는데 크게 무리가 없으나 실제 다른 게임들은 경우의 수가 무수히 많다. 만약 한가지 상태에서 10가지 경우가 나온다면, 3수 앞을 내다보는 인공지능은 1000가지 상태를, 6수 앞을 내다보는 인공지능은 1,000,000가지 상태를 평가해야한다. 그래서 저 연산 중 불필요한 연산을 가지치듯이 버리는 알고리즘을 alpha/beta pruning(알파 베타 가지치기) 라고 한다.

alpha/beta pruning알고리즘을 그림으로 살펴보면 다음과 같다.



자신이 그 경우를 택하면 자신이 불리해지는 것이 확정된 경우, 어떠한 경우가 자신에게 유리한 것이 확정되어 상대가 그것을 택하지 않을 확률이 높은 경우, 이 두가지 경우는 애써 모든 자식노드를 검사해봐야 선택하지 않을 확률이 매우 높다. 그래서 저러한 경우가 확정되면 그 노드의 자식노드에 대한 평가를 당장 중지하여야만 불필요한 연산을 줄일 수 있다.

**Alpha cut-off**는 자신이 상대방보다 불리하여, 자신이 그 경우를 선택하지 않을 때 불필요한 연산을 잘라내는 것이고, **Beta cut-off** 는 자신이 상대방보다 유리하여, 상대방이 그 경우를 선택하지 않을 확률이 높을 때 불필요한 연산을 잘라내는 것이다.

2) **몬테카를로 알고리즘**

① 몬테카를로 알고리즘은 수학적 계산을 통해 정확한 결과값을 얻는 방 법이 아니라 랜덤한 숫자를 생성시켜 어떤 문제의 답을 확률적으로 예측 하는 것이다. 예를 들어 원의 넓이를 구하고 싶다고 치자. 그러면 원과 원 을 둘러싸는 정사각형을 그리고 그 위에 임의의 점들을 막 생성해보자. 그 러고 나서 전체 찍힌 점들 중 원 위에 찍힌 점들의 비율을 구하면 확률적 인 방법으로 원의 넓이를 대략적으로 구할 수 있게 되는 것이다.

② 아래 파이썬 코드는 전체 점의 개수 dot\_cnt을 증가시킴으로써 실제 원의 넓이인 pi와 값이 비슷해지는 것을 볼 수 있다.

import math

import random

dot\_cnt = 10 # dot\_cnt 를 10, 100, 1000, 10000, 100000, 1000000로

#늘려가며 원의 넓이가 pi(3.141592…)와 가까워지는지 보자.

cnt\_in = 0

for i in range(dot\_cnt):

x = random.uniform(-1,1)

y = random.uniform(-1,1)

if math.pow(x,2)+math.pow(y,2) <= 1:

cnt\_in += 1

print('원의 넓이 : ')

print((cnt\_in/dot\_cnt)\*4)

③ 몬테카를로 알고리즘이 필요한 이유

많은 요소들이 작용하는 불확실한 상황에서 의사결정을 하기 위함이다.

랜덤 값을 발생시켜 시뮬레이션을 함으로써 직접 계산하기 어려운 복잡한 수치들을 구할 필요가 있을 때 유용하다. 실제 세상에서 발생되는 거의

모든 일들은 확률로 설명될 수 있다. 이러한 확률적 현상들을 수학적으로 풀고 싶은데 그것을 수학적으로 정밀하게 풀어낼 수 없을 때 랜덤 값을 발생시켜 시뮬레이션함으로서 정답에 근접한 해를 구할 수 있는 것이다.

※ 일반학습 중 min/max 알고리즘을 적용하여 TIC TAC TOE 프로그램을 돌리는데 필요한 기능

1) printboard 함수 : 보드판을 그리는 기능

2) emptystate 함수 : 보드판을 초기화 하는 기능

3) gameover 함수 : 게임이 끝났는지를 체크하는 기능

4) [create\_children 함수](#create_children함수) : 노드를 생성하는 기능

5) [checkval 함수](#checkval함수) : heuristics value를 출력하는 기능

(depth 0 에 출력되는 값, 이기면 1 지면 -1 비기면 0)

6) [minmax 함수](#minmax함수) : 만들어진 노드를 검색해서 재귀를 이용하여 값을 역전파 시켜주는 기능

7) play 함수 : 게임을 진행하는 기능

8) action 함수 : 다음 수를 출력하는 기능

9) episode\_over 함수 : 누가 이겼는지를 출력하는 기능

나머지는 tic tac toe 기본 코드에서 살펴봤던 함수들이고 이 중 4~6번 함수가 min/max 알고리즘을 구현하는 함수 이므로 이들을 살펴보자.

1) **checkval 함수** : root node 의 상태(p)를 출력하는 기,능gameover 함수와 비슷하다.(gameover 함수에서 return 값만 100,-100,0,-2로 바꾸어 준다.)

def checkval(state):

#from sys import maxsize

maxsize = 100

DRAW = -2

for i in range(3):

if state[i][0] != EMPTY and state[i][0] == state[i][1] and state[i][0] == state[i][2]:

return maxsize if state[i][0] == 1 else -maxsize

if state[0][i] != EMPTY and state[0][i] == state[1][i] and state[0][i] == state[2][i]:

return maxsize if state[i][0] == 1 else -maxsize

if state[0][0] != EMPTY and state[0][0] == state[1][1] and state[0][0] == state[2][2]:

return maxsize if state[0][0] == 1 else -maxsize

if state[0][2] != EMPTY and state[0][2] == state[1][1] and state[0][2] == state[2][0]:

return maxsize if state[0][2] == 1 else -maxsize

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

return EMPTY

return DRAW

2) **create\_children 함수** : 사람이 둔 수를 기준으로 남은 수를 가지고 min/max 트리 생성하는 (재귀)함수

- create\_children 함수 만들기

def createChildren(self): #노드 생성하는 함수

global switch\_map

if abs(self.value) == maxsize:

# 만약에 판이 끝났다면 더이상 노드를 만드는 작업을 진행하지 마라.

return

if self.depth >= 0: # depth 가 0이 될때까지 루프를 돈다.

for i in range(9):

move=switch\_map[i+1] # (0,0) ,(0,1)...(2,2)

# 사람이 첫수를 (0,0) 자리에 뒀다면 상황이 [[1,0,0],[0,0,0],[0,0,0]] 이다.

if self.state[move[0]][move[1]] == 0:

# 이 밑의 코드 : 사람이 둔자리 이후 나머지 자리를 둬보며 트리를 그리는 코드

# 0 1

child\_state = deepcopy(self.state)

# deepcopy:안쪽에 있는 리스트까지 전부 카피하겠다

#위의 상황인 [[1,0,0],[0,0,0],[0,0,0]]를 child\_state에 입력

child\_state[move[0]][move[1]] = self.player\_num

# (0,1) 자리에 -1이 들어간다.

# 0 1 - 1

# 결과 : [[1,-1,0],[0,0,0],[0,0,0]]

self.children.append(Node(self.depth - 1, # depth 7

# Node 라는 클래스가 함수안에 들어와 재귀되고 있다.

-self.player\_num, # -(-1)

#-1로 할까 2로 할까

child\_state,

# [[1,-1,0],[0,0,0],[0,0,0]]

self.checkVal(child\_state),

# 그 판을 보고 이겼는지 졌는지 비겼는지 진행중인지 상황을 리턴해줌

i))

3) **minmax 함수** : 만들어진 노드를 탐색하면서 값을 역전파 시켜주는 (재귀)함수

- min/max 함수 만들기

def MinMax(node, depth, player\_num):

global switch\_map

if depth == 0 or abs(node.value) == maxsize:

# 맨아래노드에 도달했거나 맨아래 노드에 도달하지 않았다고 하더라도 누군가 이겼을 때의 그 value(100,-100)를 리턴하라

return node.value

z\_count = 0

for i in range(9):

move=switch\_map[i+1] # (0,0),(0,1),...,(2,2)

if node.state[move[0]][move[1]] == 0:

# 게임판에 빈 곳이 있으면 z\_count 를 1을 늘린다.

z\_count += 1

if z\_count == 0:

# 다 채웠는데 비겼다면 -2를 리턴하라 (이기거나 졌으면 앞의 if 문에서 결과가 나왔다)

# ex) [[1,-1,-1],[-1,1,1],[1,-1,-1]]과 같은 상황인 경우 : z\_count == 0 인 경우 이다.

return -2

best\_value = maxsize \* -player\_num

# -100 = 100 \* - 1

for i in range(len(node.children)):

# 수를 둘때마다 현 상태에서 나머지 판을 다 돌려서

# ex : depth = 2 일때의 node에 child node 가 2개라서 2번 루프를 도는 상황

child = node.children[i]

# 지금 상황에 best value를 돌려주는 부분

value = MinMax(child, depth - 1, -player\_num)

# 첫번째 루프 : i = 0 일 때 value 에 100이 들어있음

## 두번째 루프 : i = 1 일 때 value 에 -2가 들어있음

# value = 100, maxsize = 100, best\_value = -100, player\_num = 1

## value = 100, maxsize = 100, best\_value = 100, player\_num = 1

if abs(maxsize \* player\_num - value) < abs(maxsize \* player\_num - best\_value):

# 100 \* 1 - 100 < 100 \* 1 - (-100)

# 0(abs : 절대값 계산해주는 메소드) < 200 --> true

## 100 \* 1 - (-2) < 100 \* 1 - 100

## 102 < 0 --> false

best\_value = value

# 100

## 두번째 루프에서는 false 이기 때문에 best\_value 값이 업데이트 되지 못한다.

return best\_value

# 따라서 depth = 2 node 에서의 best\_value 값으로 100을 가져온 것이다.

(3) 강화학습을 적용한 TIC TAC TOE 코드

※ TIC TAC TOE 프로그램에 강화학습을 적용하기 위해서 필요한 기능

1) [readCSV](#readCSV) : 학습데이터를 읽어오는 함수

2) [statetuple](#statetuple) : 리스트를 튜플로 변환해주는 함수

3) [lookup](#lookup) : 현재판을 기존 학습 데이터와 비교하는 함수

4) [add](#add) : 한수 한수 둘 때마다 데이터를 만드는 함수

5) [winnerval](#winnerval) : 보상함수(1,0,-1,0.5점)

6) [greedy](#greedy) : 다음 수 중에 가장 좋은 수가 어디인지 출력해주는 함수

7) [backup](#backup) : 보상에 따라서 내가 뒀던 이전 수에 가중치를 갱신해주는 함수

8) [random](#random) : 새로운 수를 둬 좋은 수를 알아내기 위해 랜덤 수를 두는 함수

1) **readCSV** : 학습데이터를 읽어오는 함수

(backup 함수로 학습데이터를 생성하기 전 미리 학습된 데이터를 가져오는 함수)

- 이미 학습된 데이터(수,가중치) ttt\_learn\_data.csv를 읽어와서 빈딕셔너리에 입력하는 함수 생성

def readCSV():

import csv

dic = {}

file = open("d://data//ttt\_learn\_data.csv","r")

ttt\_list = csv.reader(file)

for t in ttt\_list:

try:

dic[ ( ( int(t[0]), int(t[1]), int(t[2]) ),

( int(t[3]), int(t[4]), int(t[5]) ),

( int(t[6]), int(t[7]), int(t[8]) ) ) ] = float(t[10])

except ValueError: # 데이터 안에 있는 컬럼 때문에 생기는 오류 해결

continue

return dic

print(readCSV())

2) **statetuple** : 리스트를 튜플로 변환해주는 함수

* 아래의 state 리스트를 튜플로 만들어질수 있도록 하는 함수를 생성

state = [[1,1,2],[1,2,1],[2,2,1]]

i)

def statetuple(state):

return (tuple(state[0]),tuple(state[1]),tuple(state[2]))

ii)

def stateTuple(state) :

res = []

for i in state :

res.append(tuple(i))

return tuple(res)

iii)

def statetuple(state):

return tuple(map(tuple, state))

state = [[1,1,2],[1,2,1],[2,2,1]]

print(statetuple(state))

결과 : ((1, 1, 2), (1, 2, 1), (2, 2, 1))

3) **lookup** : 가장 좋은 수를 알아내기 위해서 남아있는 다음 수의 판이 그동안 학습된 데이터에 있는지 찾아보고 있으면 바로 그 가중치를 출력하고 없으면 새로 추가하고 그 가중치를 출력하는 함수

(수를 둘 때마다 기본적으로 0.5의 가중치를 둔다.)

ex) 현재수 ---------------> 다음수 가중치

(1,2,0)(0,0,0)(0,0,0) (1,2,1)(0,0,0)(0,0,0) 0.5

(1,2,0)(1,0,0)(0,0,0) 0.7

(1,2,0)(0,1,0)(0,0,0) 0.6

(1,2,0)(0,0,1)(0,0,0) 0.4

(1,2,0)(0,0,0)(1,0,0) 0.2

(1,2,0)(0,0,0)(0,1,0) 0.3

(1,2,0)(0,0,0)(0,0,1) 0.1

수를 둘 때마다 기본적으로 0.5의 가중치를 주는데 만약 이 수로 승리를 했다면 가중치를 갱신해준다.

* [[1,1,2],[1,2,0],[2,0,0]] 라는 수가 학습된 데이터에 있는지 확인하여 있다면 그 판을 출력하고 없으면 없다는 메시지가 출력되는 lookup함수 만들기

def lookup(state):

key = statetuple(state)

values = readCSV()

if key not in values:

return "없습니다"

else:

return key

state = [[1,1,2],[1,2,0],[2,0,0]]

print(lookup(state))

4) **winnerval** : 보상해주는 함수, 이기면 1점을 보상해주고 지면 -1 점을 감점하고 비기면 0점을 주고 게임이 진행 중이면 0.5점을 준다.

" agent 가 tic tac toe 게임에 적응해 나가기 위한 함수이다"

winnerval 함수 생성 : 숫자가 1,2 가 입력되면 1이 출력

숫자가 0 이 입력되면 0.5가 출력

숫자가 3이 입력되면 0이 출력

그 밖에 숫자가 입력되면 -1 출력

def winnerval(winner):

if winner in (1,2):

return 1

elif winner == 0:

return 0.5

elif winner == 3:

return 0

else:

return -1

winnerval(3)

5) **add** : lookup을 통해서 학습된 데이터를 찾았으면 이 함수는 사용되지 않는다. 이 함수는 lookup을 통해서 그 다음에 두어야 할 수를 찾지 못했을 때 새로운 수를 추가하는 함수이다.

ex) 현재수 ---------------> 다음수 가중치

(1,2,0)(0,0,0)(0,0,0) (1,2,1)(0,0,0)(0,0,0) 0.5

(1,2,0)(1,0,0)(0,0,0) 0.7

(1,2,0)(0,1,0)(0,0,0) 0.6

학습된 데이터 중에 있는 데이터

--------------------------------------------------------------------------

학습된 데이터 중에 없는 데이터

(1,2,0)(0,0,1)(0,0,0) 0.5

(1,2,0)(0,0,0)(1,0,0) 0.5

(1,2,0)(0,0,0)(0,1,0) 0.5

(1,2,0)(0,0,0)(0,0,1) 0.5

내가 이 상황에서 다음 수를 둘 때 학습된 데이터는 이기기도 하고 져보기도 했기 때문에 0.5에서 가중치가 변하였다. 그러나 한번도 안 두었던 데이터의 경우에는 0.5 가중치로 처음에 추가가 되는데 이 때 add 함수로 추가한다.

* add 함수 만들기

def add(state):

values ={}

tup = statetuple(state)

winner = gameover(state)

values[tup] = winnerval(winner)

return values

state = [[1,1,2],[1,2,0],[0,0,0]]

print(add(state))

6) **greedy** : 현재 수 이후에 다음 수 중에 가장 좋은 수가 어디인지 출력해주는 함수, 가장 좋은 수를 찾기 위해서 lookup 함수를 통해서 학습된 데이터 중 가중치를 받아와 가중치가 가장 큰 수를 찾아낸다.

만약 ((1,1,2),(2,1,2),(0,0,0)) 이러한 상황에서 1이 수를 둘 차례인데 (2,0), (2,1), (2,2) 자리에 다 숫자 1을 둔 상태를 학습 데이터에서 있는 지를 찾아서 없으면 add를 하고 있으면 가중치를 가져와보고 그 중에서 가장 높은 수를 선택하여 리턴해주는 역할을 한다.

* greed 함수 생성

state = [[1,1,2],[2,1,2],[0,0,0]]

    def greedy(state):

        maxval = -50000

# 남아있는 수중에 가장 좋은 수의 가중치를 담기 위해  선언

        maxmove = None

# 남아있는 수중에 가장 좋은 수를 담기 위해 선언 (2,2) 같은것

        for i in range(3):

            for j in range(3):

                if state[i][j] == EMPTY: # 남아있는 수중에 비어있는 수를 찾아서

                    state[i][j] = 1 # 거기에 플레이어의 숫자를 넣은 후

                    val = lookup(state) # 그 기보의 가중치를 가져온다.

                    state[i][j] = EMPTY

# 그 수를 다시 비워준다.(다른 수의 가중치도 파악하기 위해)

                    if val > maxval:

                        maxval = val

                        # print (maxval) # 남아있는 수중에 가장 큰 가중치

                        maxmove = (i, j)

                        #print(maxmove)

 # 남아있는 수중에 가장 가중치가 큰 자리 (2,1)

        return maxmove

7) **backup** : 보상에 따라서 내가 뒀던 이전 수에 가중치를 갱신해주는 함수

기존에  readCSV 함수로 강화 학습한 정보를 불러와서 강화학습 된 tic tac toe 게임을 진행했다면 backup함수를 통해서 직접 강화학습이 되어가는 과정에 대한 정보를 만들게 된다.

이전 수의 가중치 = 가중치 + 0.99 \* 가중치 변화량

self.values[self.prevstate]+= self.alpha \* (nextval - self.prescore)

0.99 \* (0.6 - 0.5) + 0.5

(0.99 : 하이퍼파라미터)

현재수----------------------> 다음수 -----------------> 다음수

(1,2,2)(2,1,1)(1,0,0) (1,2,2)(2,1,1)(1,2,0) (1,2,2)(2,1,1)(1,2,1)

0.5 ->0.9901 갱신 0.5 ->0.995로 갱신 1 승리

이기기 위해 가장 중요했던 전수의 가중치를 업데이트 해준다.

0.5 + 0.99 \* (1-0.5) = 0.995

0.5 + 0.99 \*(0.995-0.5) = 0.9901

이런 식으로 가중치가 역전파가 된다. 이렇게 역전파가 되면 될수록 가중치의 값이 낮아진다. 바둑이나 오목과 같이 처음에는 어디에 두어도 상관없다면 이렇게 해도 되지만 첫수부터 중요한 장기 같은 경우에는 이렇게 적용하면 학습이 느리게 되거나 학습이 되지 않을 수도 있다.

- backup 함수를 생성하여 학습데이터에 가중치가 갱신되도록 만들기

def backup(self,nextval):

if self.prevstate != None and self.learning:

self.values[self.prevstate] += self.alpha \* (nextval - self.prevscore)

- 위에서 만든 backup 함수를 실행시켜주어야 하는 두 곳

게임의 승패가 결정되는 episode\_over 함수

매 수마다 최적의 수를 찾는 greedy 함수

- backup 함수에서 입력하고 있는 변수에 대한 정의는 init 함수에서 내리기.

def \_\_init\_\_(self, player, learning = True):

self.values = {}

self.learning = learning # learning의 default값을 True로 해놓았다.

self.alpha = 0.99 # 감가율 (하이퍼파라미터)

self.prevstate = None # 이전에 두었던 수를 위한 변수

self.prevscore = 0 # 이전에 두었던 수에 대한 가중치 저장 변수

8) **random** : 새로운 탐험을 통해서 좋은 수를 알아내기 위해 랜덤 수를 던지기 위한 함수(비율을 잘 조정해주어야 한다.)

강화 학습이 되기 위해서는 agent가 탐색(exploit)과 탐험(explore)을 해야하는데 탐색은 이전 경험에서 최적의 수를 찾는 것이고 탐험은 새로운 모험(임의의 수)을 통해서 최적의 수를 찾는 것이다. 탐색을 하기 위해서 greedy 알고리즘을 사용하게 되고 탐험을 위해 random 함수를 통해 새로운 모험을 하게 한다. 이렇게 탐색과 탐험을 같이하는 알고리즘을 ε - greedy 알고리즘이라고 한다. (ε : epsilon)

- ε - greedy 알고리즘이 되도록 탐색을 하는 과정에 탐험하는 코드 작성

computer class 의 \_\_init\_\_ 에 추가

def \_\_init\_\_(self,player):

self.player = player

self.epsilon = 0.1 # 10번 수를 둔다면 1번은 탐험을 하겠다.

computer class 의 action 에 추가

def action(self,state):

printboard(state)

r = random.random()

if r < self.epsilon:

move = self.random(state)

else:

move = self.greedy(state)

state[move[0]][move[1]] = self.player

return move

- 컴퓨터가 스스로 자기 자신과 tic tac toe 게임을 10000번 하게 하고 그렇게 학습한 데이터를 바탕으로 사람과 게임을 하게하는 최종코드 작성

import random, csv

from copy import copy, deepcopy

EMPTY = 0

PLAYER\_X = 1

PLAYER\_O = 2

DRAW = 3

BOARD\_FORMAT = "----------------------------\n"\

"| {0} | {1} | {2} |\n"\

"|--------------------------|\n"\

"| {3} | {4} | {5} |\n"\

"|--------------------------|\n"\

"| {6} | {7} | {8} |\n"\

"----------------------------"

NAMES = [' ', 'X', 'O']

def printboard(state):

cells = []

for i in range(3):

for j in range(3):

cells.append(NAMES[state[i][j]].center(6))

print(BOARD\_FORMAT.format(\*cells))

def emptystate():

return [[EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY]]

def gameover(state):

for i in range(3):

if state[i][0] != EMPTY and state[i][0] == state[i][1] and state[i][0] == state[i][2]:

return state[i][0]

if state[0][i] != EMPTY and state[0][i] == state[1][i] and state[0][i] == state[2][i]:

return state[0][i]

if state[0][0] != EMPTY and state[0][0] == state[1][1] and state[0][0] == state[2][2]:

return state[0][0]

if state[0][2] != EMPTY and state[0][2] == state[1][1] and state[0][2] == state[2][0]:

return state[0][2]

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

return EMPTY

return DRAW

class Human(object):

def \_\_init\_\_(self, player):

self.player = player

def action(self, state):

printboard(state)

action = None

switch\_map = {1: (0, 0), 2: (0, 1), 3: (0, 2),

         4: (1, 0), 5: (1, 1), 6: (1, 2),

         7: (2, 0), 8: (2, 1), 9: (2, 2) }

cycle = True

while action not in range(1, 10) or cycle:

try :

action = int(input('Your move? '))

if state[switch\_map[action][0]][switch\_map[action][1]] != 0 :

print('이미 둔 수입니다.')

else :

cycle = False

except ValueError :

print('허용값이 아닙니다. 다시 입력하세요.')

continue

except KeyError :

print('허용범위를 초과하였습니다. 다시 입력하세요.')

continue

return switch\_map[action]

def episode\_over(self, winner):

if winner == DRAW:

print('Game over! It was a draw.')

else:

print('Game over! Winner: Player {0}'.format(winner))

def play(agent1, agent2):

state = emptystate()

for i in range(9):

if i % 2 == 0:

move = agent1.action(state)

else:

move = agent2.action(state)

state[move[0]][move[1]] = (i % 2) + 1

winner = gameover(state)

if winner != EMPTY:

return winner

return winner

class Computer(object):

def \_\_init\_\_(self, player, lossval=-1, learning = True):

self.player = player

self.dic = {}

self.lossval = lossval

self.epsilon = 0.1

self.learning = learning

self.alpha = 0.99

self.prevstate = None

self.prevscore = 0

def random(self, state):

available = []

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

available.append((i ,j))

return random.choice(available)

def greedy(self, state):

maxval = -50000

maxmove = None

for i in range(3):

for j in range(3):

if state[i][j] == EMPTY:

state[i][j] = self.player

val = self.lookup(state)

state[i][j] = EMPTY

if val > maxval:

maxval = val

maxmove = (i, j)

self.backup(maxval)

return maxmove

def lookup(self, state):

key = self.statetuple(state)

if not key in self.dic:

self.add(key)

return self.dic[key]

def add(self, state):

winner = gameover(state)

tup = self.statetuple(state)

self.dic[tup] = self.winnerval(winner)

def statetuple(self,state):

return tuple(map(tuple, state))

def action(self, state):

#printboard(state)

r = random.random()

if r < self.epsilon:

move = self.random(state)

else:

move = self.greedy(state)

state[move[0]][move[1]] = self.player

self.prevstate = self.statetuple(state)

self.prevscore = self.lookup(state)

return move

def backup(self,nextval):

if self.prevstate != None and self.learning:

self.dic[self.prevstate] += self.alpha \* (nextval - self.prevscore)

def winnerval(self, winner):

if winner == self.player:

return 1

elif winner == EMPTY:

return 0.5

elif winner == DRAW:

return 0

else:

return self.lossval

def episode\_over(self, winner):

self.backup(self.winnerval(winner)) # 한판 끝나고 결과 가중치 정보 갱신

self.prevstate = None # 실행문에서 Computer clas를 한번만 호출하는데

self.prevscore = 0 # 매판마다 가중치 정보 갱신하고 값을 초기화 해주어야 하기 때문에 함수 안에서 두 변수를 다시 초기화 해준다.

if \_\_name\_\_ == "\_\_main\_\_":

p1 = Computer(1)

p2 = Computer(2)

for i in range(10000):

if i % 10 == 0:

print('Game: {0}'.format(i))

winner = play(p1, p2)

p1.episode\_over(winner)

p2.episode\_over(winner)

while True:

p1 = Human(1)

winner = play(p1, p2)

p1.episode\_over(winner)

p2.episode\_over(winner)

- 4목 틱텍토 코드

import random

from copy import copy, deepcopy

EMPTY = 0

PLAYER\_X = 1

PLAYER\_O = 2

DRAW = 3

BOARD\_FORMAT = "------------------------------------\n"\

"| {0} | {1} | {2} | {3} |\n"\

"|-----------------------------------|\n"\

"| {4} | {5} | {6} | {7} |\n"\

"|-----------------------------------|\n"\

"| {8} | {9} | {10} | {11} |\n"\

"|-----------------------------------|\n"\

"| {12} | {13} | {14} | {15} |\n"\

"------------------------------------"

NAMES = [' ', '●', '○']

def printboard(state):

cells = []

for i in range(4):

for j in range(4):

cells.append(NAMES[state[i][j]].center(6))

print(BOARD\_FORMAT.format(\*cells))

def emptystate():

return [[EMPTY, EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY, EMPTY]]

def gameover(state):

for i in range(4):

if state[i][0] != EMPTY and state[i][0] == state[i][1] and state[i][0] == state[i][2] and state[i][0] == state[i][3]:

return state[i][0]

if state[0][i] != EMPTY and state[0][i] == state[1][i] and state[0][i] == state[2][i] and state[0][i] == state[3][i]:

return state[0][i]

if state[0][0] != EMPTY and state[0][0] == state[1][1] and state[0][0] == state[2][2] and state[0][0] == state[3][3]:

return state[0][0]

if state[0][3] != EMPTY and state[0][3] == state[1][2] and state[0][3] == state[2][1] and state[0][3] == state[3][0] :

return state[0][3]

for i in range(4):

for j in range(4):

if state[i][j] == EMPTY:

return EMPTY

return DRAW

class Computer(object):

def \_\_init\_\_(self, player, lossval=-1, learning=True):

self.dic = {}

self.player = player

self.lossval = lossval

self.learning = learning

self.epsilon = 0.1

self.alpha = 0.99

self.prevstate = None

self.prevscore = 0

def episode\_over(self, winner):

self.backup(self.winnerval(winner))

self.prevstate = None

self.prevscore = 0

def action(self, state):

r = random.random()

if r < self.epsilon:

move = self.random(state)

else:

move = self.greedy(state)

state[move[0]][move[1]] = self.player

self.prevstate = self.statetuple(state)

self.prevscore = self.lookup(state)

state[move[0]][move[1]] = EMPTY

return move

def random(self, state):

available = []

for i in range(4):

for j in range(4):

if state[i][j] == EMPTY:

available.append((i, j))

return random.choice(available)

def greedy(self, state):

maxval = -50000

maxmove = None

for i in range(4):

for j in range(4):

if state[i][j] == EMPTY:

state[i][j] = self.player

val = self.lookup(state)

state[i][j] = EMPTY

if val > maxval:

maxval = val

maxmove = (i, j)

self.backup(maxval)

return maxmove

def backup(self, nextval):

if self.prevstate != None and self.learning:

self.dic[self.prevstate] += self.alpha \* (nextval - self.prevscore)

def lookup(self, state):

key = self.statetuple(state)

if not key in self.dic:

self.add(key)

return self.dic[key]

def add(self, state):

winner = gameover(state)

tup = self.statetuple(state)

self.dic[tup] = self.winnerval(winner)

def winnerval(self, winner):

if winner == self.player:

return 1

elif winner == EMPTY:

return 0.5

elif winner == DRAW:

return 0

else:

return self.lossval

def statetuple(self,state):

return tuple(map(tuple, state))

class Human(object):

def \_\_init\_\_(self, player):

self.player = player

def action(self, state):

printboard(state)

action = None

switch\_map = {1: (0, 0), 2: (0, 1), 3: (0, 2), 4: (0, 3),

5: (1, 0), 6: (1, 1), 7: (1, 2), 8: (1, 3),

9: (2, 0),10: (2, 1),11: (2, 2),12: (2, 3),

13: (3, 0),14: (3, 1),15: (3, 2),16: (3, 3)}

cycle = True

while action not in range(1, 17) or cycle:

try :

action = int(input('Your move? '))

if state[switch\_map[action][0]][switch\_map[action][1]] != 0 :

print('이미 둔 수입니다.')

else :

cycle = False

except ValueError :

print('허용값이 아닙니다. 다시 입력하세요.')

continue

except KeyError :

print('허용범위를 초과하였습니다. 다시 입력하세요.')

continue

return switch\_map[action]

def episode\_over(self, winner):

if winner == DRAW:

print('Game over! It was a draw.')

else:

print('Game over! Winner: Player {0}'.format(winner))

def play(agent1, agent2):

state = emptystate()

for i in range(16):

if i % 2 == 0:

move = agent1.action(state)

else:

move = agent2.action(state)

state[move[0]][move[1]] = (i % 2) + 1

winner = gameover(state)

if winner != EMPTY:

return winner

return winner

if \_\_name\_\_ == "\_\_main\_\_":

p1 = Computer(1)

p2 = Computer(2)

for i in range(100000):

if i % 10 == 0:

print('Game: {0}'.format(i))

winner = play(p1, p2)

p1.episode\_over(winner)

p2.episode\_over(winner)

while True:

p1 = Human(1)

winner = play(p1, p2)

p1.episode\_over(winner)

5. 강화 학습을 구현한 게임 2 - Ping Pong

(1) Ping Pong 기본코드 이해

(2) 강화학습을 적용한 Ping Pong 코드

(1) Ping Pong 기본코드 이해

※ 2개의 class로 이루어진 Ping Pong 코드 각각의 기능

o Ball Class

1) [init함수](#ball_init함수)

2) [draw함수](#ball_draw함수) : 공을 움직이게 하는 함수

3) [hit\_paddle함수](#hit_paddle함수) : 공이 패들에 튀기게 하는 함수

o Paddle Class

1) [init함수](#paddle_init함수)

2) [draw함수](#paddle_draw함수) : 패들을 움직이게 하는 함수

3) [turn\_left, turn\_right함수](#turn_left함수) : 패들을 좌우로 움직이는 함수

o [Ping Pong game 실행문](#pingpong실행문)

1) Ball Class의 **init함수**

def \_\_init\_\_(self, canvas, paddle, color):

self.canvas = canvas

self.paddle = paddle #공의 좌표 : 서,남,동,북, 공의 색깔

self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)

#oval : object형태 타입

self.canvas.move(self.id, 245, 100) # 공을 캔버스 중앙으로 이동

starts = [-3, -2, -1, 1, 2, 3] #공의 방향을 랜덤으로 구성하기 위한 리스트

random.shuffle(starts) # starts 리스트 중에 공이 향할 방향을 임의 지정

self.x = starts[0] # 공의 좌우 방향 시작지점을 임의로 지정

self.y = -3 # 랜덤으로 정한 x좌표 방향 위로 올라가는 속도

self.canvas\_height = self.canvas.winfo\_height()# 캔버스 현재 높이 반환

self.canvas\_width = self.canvas.winfo\_width() # 캔버스 현재 넓이 반환

#self.hit\_bottom = False #공이 화면에서 사라지지 않게 하기 위해

# 바닥에 닿으면 게임 끝나는 코드를 구현하기 위해서 쓰는 변수

2) Ball Class의 **draw함수** : 공을 움직이게 하는 함수

공의 좌표 = [10,10,25,25] (공의 좌측 상단x,y , 공의 우측하단 x,y)



① 공이 위로 올라가서 천장에 부딪치면 스스로 아래로 튀기게 하는 코드

------------------------------------------------------- 천정 : 이 위치를 0으로 보고

↑

●

if pos[1] <= 0:

self.y = 3

② 공이 아래로 내려가서 바닥에 부딪치면 스스로 위로 튀기게 하는 코드

●

↓

--------------------------------------------------- 바닥 : 이 위치는canvas\_height

if pos[3] >= self.canvas\_height:

self.y = -3

③ 공이 왼쪽으로 가서 벽에 부딪치면 다시 오른쪽으로 튀기게 하는 코드

|

|

| ← ●

|

|

if pos[0] <= 0:

self.x = 3

④ 공이 오른쪽으로 가서 벽에 부딪치면 다시 왼쪽으로 튀기게 하는 코드

|

|

● →|

|

|

if pos[2] <= self.canvas\_width:

self.x = -3

- draw 함수 전체 코드

def draw(self):

self.canvas.move(self.id,self.x,self.y)

pos = self.canvas.coords(self.id)

if pos[1] <= 0:

self.y = 3

if pos[3] >= self.canvas\_height:

self.y = -3

if pos[0] <= 0:

self.x = 3

if pos[2] >= self.canvas\_width:

self.x = -3

3) **hit\_paddle함수** : 공이 패들에 튀기게 하는 함수

def hit\_paddle(self,pos):

paddle\_pos = self.canvas.coords(self.paddle.id)

if pos[2] >= paddle\_pos[0] and pos[0] <= paddle\_pos[2]: # 좌,우 확인

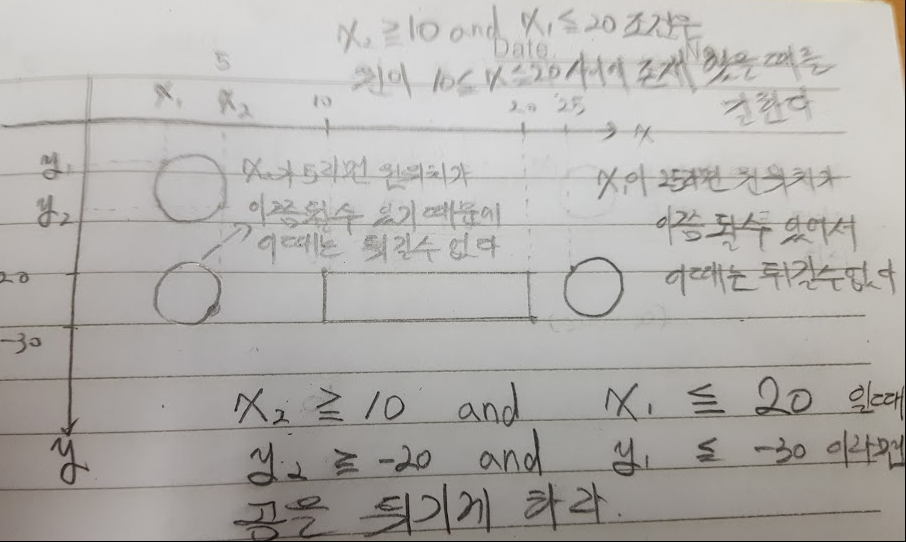
if pos[3] >= paddle\_pos[1] and pos[1] <= paddle\_pos[3]:

return True # 위,아래 확인

return False

코드 설명 :

공의 우측하단 x좌표가 패들의 좌측상단 x좌표와 같거나 크고 공의 좌측상단 x좌표가 패들의 우측하단 x좌표와 같거나 작을 경우에 공의 우측하단 y좌표가 패들의 좌측상단 y좌표와 크거나 같고 공의 좌측상단 y좌표가 패들의 우측하단 y좌표와 같거나 작을 경우 True를 리턴하고 그 이외의 경우에는 False를 리턴.



공이 앞의 if 조건에 위치할 때 y좌표는 사실상 -30으로 올 수는 없기 때문에 굳이 없어도 상관은 없다라고 생각할 수 있으나 지금 이 코드에서 정의한 원의 두 점은 각이 져 있는 것이 아니기 때문에 미세한 차이로 앞의 if조건 밖에서도 튕겨야 할 상황이 올 수 있기 때문에 pos[1] <= paddle\_pos[3] 까지 주는 것이다.

4) Paddle class의 **init함수**

def \_\_init\_\_(self,canvas,color):

self.canvas = canvas

self.id = canvas.create\_rectangle(0,0,100,10,fill=color)

# 패들의 높이와 넓이 그리고 색깔

self.canvas.move(self.id, 200, 400) # 패들 사각형을 200,400에 위치함

self.x = 0 # 패들이 처음 시작할 때 움직이지 않게 0으로 설정

self.y = 0

self.canvas\_width = self.canvas.winfo\_width()

self.canvas\_height = self.canvas.winfo\_height()

# 캔버스 밖으로 패들이 나가지 않도록 캔버스 높이, 넓이 반환

self.canvas.bind\_all('<KeyPress-Left>',self.turn\_left)

self.canvas.bind\_all('<KeyPress-Right>',self.turn\_right)

# 수동조작을 원할 때 패들 이동키 설정법

5) Paddle class의 **draw함수**

- 사람이 임의로 패들을 조종할 수 있게 하는 코드

# 패들이 화면의 끝에 부딪히면 공처럼 튕기는게 아니라 움직임이 멈춰야한다.

def draw(self):

paddle\_pos = self.canvas.coords(self.id)

pos = self.canvas.coords(self.id)

if pos[0] <= 0 and self.x < 0: # 패들의 위치가 왼쪽 끝이고, 이동하려는

return # 방향이 왼쪽이면 함수 종료(이동 안 함)

elif pos[2] >= self.canvas\_width and self.x > 0: # 패들의 위치가 오른쪽 끝

return # 이고, 이동하려는 방향이 오른쪽이면 함수 종료

self.canvas.move(self.id, self.x,self.y)

- 컴퓨터가 임의로 패들을 조종할 수 있게 하는 코드

def draw(self):

self.canvas.move(self.id, self.x,self.y)

paddle\_pos = self.canvas.coords(self.id)

paddle\_move = random.random()

#패들의 움직임을 0~1사이에 임의로 지정한다.

if paddle\_move <= 0.5 and paddle\_pos[0] >= 0 :

self.turn\_left()

elif paddle\_move <= 1 and paddle\_pos[2] <= self.canvas\_width:

self.turn\_right()

임의의 0~1사이의 움직임을 반으로 잘라서 0~0.5범위에서는 패들을 왼쪽으로 움직이도록 하고 0.5~1범위에서는 패들을 오른쪽으로 움직이게 한다.

그리고 0~0.5범위에서 움직일 때 패들이 좌측 벽으로 나갈 수 없게끔 패들의 좌측 상단 x좌표가 0보다 크다는 조건을 주고 0.5~1범위에서 움직일 때 패들이 우측 벽으로 나갈 수 없게끔 패들의 우측 하단 x좌표가 캔버스의 넓이보다 작다라는 조건을 주게 된다.

6) **turn\_left, turn\_right 함수** : 패들을 좌우로 움직이는 함수

- 사람이 임의로 패들을 조종할 수 있게 하는 코드

def turn\_left(self,evt):

self.x = -9

def turn\_right(self,evt):

self.x = 9

# 사람이 임의로 조종하게 하려면 함수에 인자를 반드시 받아야 한다.

- 컴퓨터가 임의로 패들을 조종할 수 있게 하는 코드

def turn\_left(self):

self.x = -9

def turn\_right(self):

self.x = 9

# 컴퓨터가 임의로 조종하게 하려면 함수에 인자를 받으면 안된다.

7) **Ping Pong 실행문, while loop 문**

tk = Tk() # tk 를 인스턴스화 한다. (from tkinter import \*를 쓸 때 필수 작업)

tk.title("Game") # tk 객체의 title 메소드(함수)로 게임창에 제목을 부여한다.

tk.resizable(0, 0) # 게임창의 크기는 가로나 세로로 변경될 수 없다라고 정의

tk.wm\_attributes("-topmost", 1)

# 다른 모든 창들 앞에 캔버스를 가진 창이 위치할것을 tkinter 에게 알려준다.

canvas = Canvas(tk, width=600, height=500, bg="black", highlightthickness=0)

# bg=0, highlightthickness=0 은 캔버스 외곽에 둘러싼 외곽선이 없앤다.

# 게임화면이 좀더 좋게하기 위함 , bg="black" 캔버스 색깔을 black으로 변경

canvas.pack() # 앞의 코드에서 전달된 폭과 높이는 매개변수에 따라 크기를 맞추라고 캔버스에에 말해준다.

tk.update() # tkinter 에게 게임에서의 애니메이션을 위해 자신을 초기화하라고 알려주는 것이다.

paddle = Paddle(canvas,'white') # 패들 색깔을 white로 지정

ball = Ball(canvas, paddle, 'white') #공 색깔을 white로 지정

while 1:

ball.draw()

paddle.draw()

tk.update\_idletasks()

# 우리가 창을 닫으라고 할때까지 계속해서 tkinter 에게 화면을 그려라

tk.update()

time.sleep(0.01) # 무한 루프 중에 100분의 1초마다 잠들어라 !

(2) 강화학습을 적용한 Ping Pong 코드

※ Ping Pong 프로그램에 강화학습을 적용하기 위해서 필요한 기능

o Ball class

1) init 함수

2) hit\_paddle 함수

3) hit\_paddle2 함수

4) is\_paddle\_hit 함수

5) draw 함수

o Paddle class

1) init 함수

2) draw 함수

3) turn\_left, turn\_right 함수

4) move 함수

o Agent class

1) init 함수

2) action함수

3) randomChoice 함수

4) greedyChoice 함수

5) add 함수

6) lookup 함수

7) keystate 함수

8) cyclestate 함수

9) backup 함수

10) gameover 함수

11) winnerval 함수

12) writecsv, loadcsv 함수

o Ping Pong 실행문