**Vivekanand Education Society’s Institute of Technology**

**Department of AI&DS Engineering**



**Subject: Reinforcement Learning**

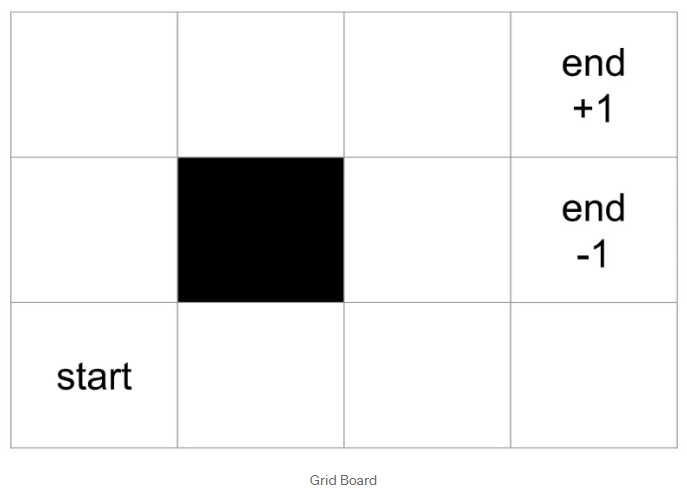
**Class: D16AD**

| ROLL NO: **30** | NAME: [**SUHANEE KANDALKAR**](mailto:2021.suhanee.kandalkar@ves.ac.in) | | |
| --- | --- | --- | --- |
| EXPERIMENT NO:**1** | TITLE:Implement a simple grid-world environment and train an agent using basic Q learning using python. | | |
| DOP: |  | DOS: **25/01/25** |  |
| GRADES: | LOs MAPPED: | | SIGNATURE: |

## Aim**:**

Implement a simple grid-world environment and train an agent using basic Q learning using python.

Theory:  
 **Grid Used**



## Code**:**

[RL\_exp1\_30.ipynb](https://colab.research.google.com/drive/1fW9xWErudi7v-Rk9aMr-NH4EVeciF3Ke#scrollTo=REkx6o--3jqs)

import numpy as np

# global variables

BOARD\_ROWS = 3

BOARD\_COLS = 4

WIN\_STATE = (0, 3)

LOSE\_STATE = (1, 3)

START = (2, 0)

DETERMINISTIC = True

class State:

def \_\_init\_\_(self, state=START):

self.board = np.zeros([BOARD\_ROWS, BOARD\_COLS])

self.board[1, 1] = -1

self.state = state

self.isEnd = False

self.determine = DETERMINISTIC

def giveReward(self):

if self.state == WIN\_STATE:

return 1

elif self.state == LOSE\_STATE:

return -1

else:

return 0

def isEndFunc(self):

if (self.state == WIN\_STATE) or (self.state == LOSE\_STATE):

self.isEnd = True

def nxtPosition(self, action):

"""

action: up, down, left, right

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

0 | 1 | 2| 3|

1 |

2 |

return next position

"""

if self.determine:

if action == "up":

nxtState = (self.state[0] - 1, self.state[1])

elif action == "down":

nxtState = (self.state[0] + 1, self.state[1])

elif action == "left":

nxtState = (self.state[0], self.state[1] - 1)

else:

nxtState = (self.state[0], self.state[1] + 1)

# if next state legal

if (nxtState[0] >= 0) and (nxtState[0] <= (BOARD\_ROWS -1)):

if (nxtState[1] >= 0) and (nxtState[1] <= (BOARD\_COLS -1)):

if nxtState != (1, 1):

return nxtState

return self.state

def showBoard(self):

self.board[self.state] = 1

for i in range(0, BOARD\_ROWS):

print('-----------------')

out = '| '

for j in range(0, BOARD\_COLS):

if self.board[i, j] == 1:

token = '\*'

if self.board[i, j] == -1:

token = 'z'

if self.board[i, j] == 0:

token = '0'

out += token + ' | '

print(out)

print('-----------------')

# Agent of player

class Agent:

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

self.states = []

self.actions = ["up", "down", "left", "right"]

self.State = State()

self.lr = 0.2

self.exp\_rate = 0.3

# initial state reward

self.state\_values = {}

for i in range(BOARD\_ROWS):

for j in range(BOARD\_COLS):

self.state\_values[(i, j)] = 0 # set initial value to 0

def chooseAction(self):

# choose action with most expected value

mx\_nxt\_reward = 0

action = ""

if np.random.uniform(0, 1) <= self.exp\_rate:

action = np.random.choice(self.actions)

else:

# greedy action

for a in self.actions:

# if the action is deterministic

nxt\_reward = self.state\_values[self.State.nxtPosition(a)]

if nxt\_reward >= mx\_nxt\_reward:

action = a

mx\_nxt\_reward = nxt\_reward

return action

def takeAction(self, action):

position = self.State.nxtPosition(action)

return State(state=position)

def reset(self):

self.states = []

self.State = State()

def play(self, rounds=10):

i = 0

while i < rounds:

# to the end of game back propagate reward

if self.State.isEnd:

# back propagate

reward = self.State.giveReward()

# explicitly assign end state to reward values

self.state\_values[self.State.state] = reward # this is optional

print("Game End Reward", reward)

for s in reversed(self.states):

reward = self.state\_values[s] + self.lr \* (reward - self.state\_values[s])

self.state\_values[s] = round(reward, 3)

self.reset()

i += 1

else:

action = self.chooseAction()

# append trace

self.states.append(self.State.nxtPosition(action))

print("current position {} action {}".format(self.State.state, action))

# by taking the action, it reaches the next state

self.State = self.takeAction(action)

# mark is end

self.State.isEndFunc()

print("nxt state", self.State.state)

print("---------------------")

def showValues(self):

for i in range(0, BOARD\_ROWS):

print('----------------------------------')

out = '| '

for j in range(0, BOARD\_COLS):

out += str(self.state\_values[(i, j)]).ljust(6) + ' | '

print(out)

print('----------------------------------')

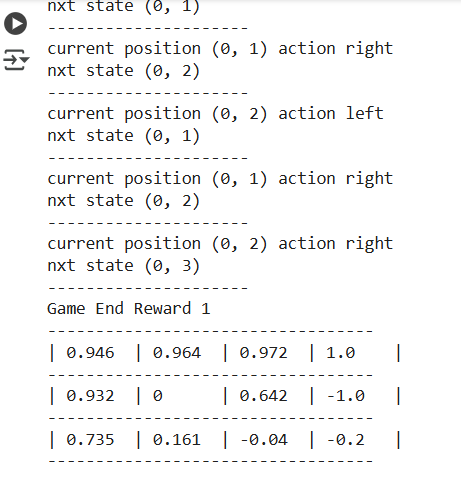
if \_\_name\_\_ == "\_\_main\_\_":

ag = Agent()

ag.play(50)

print(ag.showValues())

## Output:



# Conclusion :

In the practical implementation of the Q-learning algorithm, the agent successfully learned to navigate a grid-world environment. By updating Q-values based on rewards and utilizing an epsilon-greedy strategy for action selection, the agent gradually improved its performance. The training process allowed the agent to find an optimal path to the goal, showcasing the power of Q-learning in reinforcement learning. The implementation also highlighted the importance of parameters like learning rate, discount factor, and exploration rate in achieving efficient learning.