به نام خدا

هوم ورک چهارم درس یادگیری ماشین

مهدی فقهی

۴۰۱۷۲۲۱۳۶

با توجه به اینکه سوال اول این تمرین با بخش یادگیری تقویتی مسابقه یادگیری ماشین یکسان بود، با هماهنگی استاد درس جواب آن به صورت گروهی ارسال شده است.

گروه ما آقایان مهدی‌فقهی، محمدامین چینی فروشان و علی‌رضا صدیقی مقدم

سوال اول) در قسمت اول برای آنکه ربات ما بتواند در مقابل حریف خود مبارزه کند از الگوریتم Sarsa استفاده کردیم . برای پیاده‌سازی این الگوریتم ابتدا سعی کردیم که ربات سفید را در مقابل ربات مشکی آموزش دهیم .

قبل از شروع آموزش سعی کردیم یک دید کلی نسبت به محیط آموزش خود داشته باشیم .
با در نظر گرفتن اکشن‌ها و به علاوه اطلاعاتی که observation در اختیار ما می گذاشت متوجه شدیم که فضای حالت ما در حدود 18 * 3 * 160 * 210 * 256 حالت خواهد داشت .
سپس مجبور شدیم با دقتی بالاتر محیط را بررسی کنیم و متوجه شدیم که نتیجه‌ای که به ما برگشت داده می‌شود یک تصویر هست پس از تصویر فقط بعد صفرم را گرفتیم و محیط را به صورت سیاه و سفید در آوردیم و سپس متوجه شدیم که پیکسل‌های از تصویر نیز زائد هست و پیسکل‌های زائد تصویر را نیز حذف کردیم .
در نهایت به استراتژی رسیدیم که در آن تصویر را گرفته و موقعیت خودمان و حریفان را پیدا می کنیم سپس به عنوان حاصل دیده شده فاصله طولی و عرضی خود را با حریف در نظر می گیریم همچنین سمت و جهتی که بات ما در حال نگاه کردن است مثلا چپ ، راست یعنی ما به سمت چپ نگاه می کنیم یا راست و همچنین حریفان به چه سمتی نگاه می کند .
در نهایت به کمک این mapping جدید فضای حالت را بسیار کاهش دادیم به یک لیست چهار ورودی که ورودی اول و آخر آن دو ورودی می گیرد و ورودی سوم و چهارم نیز مقادیری بین 150- تا 150 را اختیار می کند .

```python
def map_observation_three(observation, player=1):
    # main choice result - >[position_head_first,dif_row,dif_col,position_head_second]
    # one choice result - > [position_head_first,distance,degree_line,position_head_second]

    # Capture observation
    img = observation[30:180, 30:130, 0]

    # Find player white
    first_player = img == 214
    indices_first_player = np.where(first_player)

    first_row_player_first = indices_first_player[0][0]

    first_column_player_first = indices_first_player[1][0]

    last_row_player_first = indices_first_player[0][-1]

    last_column_player_first = indices_first_player[1][-1]
```

همانطوری که در بالا می‌بینیم ما موقعیت مهره سفید را در تصویر مشخص می کنیم

```python
# Find player black
second_player = img == 0
indices_second_player = np.where(second_player)
row_first_player_second = indices_second_player[0][0]

column_first_player_second = indices_second_player[1][0]

row_last_player_second = indices_second_player[0][-1]

column_last_player_second = indices_second_player[1][-1]

head_first_pos, head_first_affine_pos, position_head_first = find_info_of_player(first_player,
                                                                                 last_row_player_first,
                                                                                 first_row_player_first,
                                                                                 last_column_player_first,
                                                                                 first_column_player_first)


head_second_pos, head_second_affine_pos, position_head_second = find_info_of_player(second_player,
                                                                                    row_last_player_second,
                                                                                    row_first_player_second,
                                                                                    column_last_player_second,
                                                                                    column_first_player_second)
```

سپس موقعیت مهره سیاه را مشخص می کنیم و با کمک این مختصات محل سر و جهتی که نگاه می کنند را پیدا می کنیم.

```
    row_head_first = head_first_pos[0]
    column_head_first = head_first_pos[1]

    row_head_second = head_second_pos[0]
    column_head_second = head_second_pos[1]

    if player == 1:
        result = np.array(
            [position_head_first, row_head_first - row_head_second, column_head_first - column_head_second,
             position_head_second])

        if row_head_first <= 31 or row_head_first >= 114 or column_head_first <= 14 or column_head_first >= 86:
            result = np.array([position_head_second, (row_head_first - row_head_second) * -1,
                               (column_head_first - column_head_second) * -1, position_head_first, row_head_first,
                               column_head_first])


    else:
        result = np.array([position_head_second, (row_head_first - row_head_second) * -1,
                           (column_head_first - column_head_second) * -1, position_head_first])

        if row_head_second <= 31 or row_head_second >= 114 or column_head_second <= 14 or column_head_second >= 86:
            result = np.array([position_head_second, (row_head_first - row_head_second) * -1,
                               (column_head_first - column_head_second) * -1, position_head_first, row_head_second,
                               column_head_second])

    return str(result.astype(int))
```

سپس به کمک اطلاعات بدست آمده، در صورتی که Player اول باشیم یا Player دوم یا سفید یا سیاه ، براساس آن فاصله طولی و عرضی را از حریف مشخص می‌کنیم در صورتی که ما در گوشه‌های رینگ قرار گرفته باشیم نیز آن اطلاعات را نیز نگه‌داری می‌کنیم .

سپس در قدم بعد سراغ پیاده‌سازی الگوریتم Sarsa در محیط boxing می‌رویم .

```
def SARSA(env, lr=0.01, num_episodes=10000, eps=0.3, gamma=0.95, eps_decay=0.00005, file_read=None, number_epoch_run=0,
          number_action=18):
    # # Initialize the Q matrix
    Q = {}
    if file_read is not None:
        with open(file_read, 'rb') as f:
            Q = pickle.load(f)
    games_reward = []
    test_rewards = []
```

یک دیکشنری به نام Q می‌سازیم که در آن حالت‌هایی که دیدیم را نگه می‌داریم و به ازای هر حالت یک لیست ۱۸ موردی اضافه می‌کنیم و به ازای اکشن‌های مختلف به آن reward انتصاب می‌دهیم و آن را به کمک فرمول Sarsa حساب می‌کنیم .

```python
for ep in range(number_epoch_run, num_episodes):
    env.reset()
    observation, reward, termination, truncation, info = env.last()
    state = map_observation_three(observation)
    done = termination or truncation
    tot_rew = 0
    # decay the epsilon value until it reaches the threshold of 0.01
    if eps > 0.01:
        eps -= eps_decay

    action = eps_greedy(Q, state, eps)

    # loop the main body until the environment stops
    # while not done:
    for agent in env.agent_iter():

        if agent == 'first_0':
            env.step(action)  # Take one step in the environment
            observation, reward, termination, truncation, info = env.last()

            next_state = map_observation_three(observation)

            if next_state not in Q.keys():
                Q[next_state] = [0 for _ in range(18)]
            done = termination or truncation
            # choose the next action (needed for the SARSA update)
            next_action = eps_greedy(Q, next_state, eps)
            reward = reward * -1
            # SARSA update
            Q[state][action] = Q[state][action] + lr * (
                    reward + gamma * Q[next_state][next_action] - Q[state][action])

            state = next_state
            action = next_action
            tot_rew += reward
```

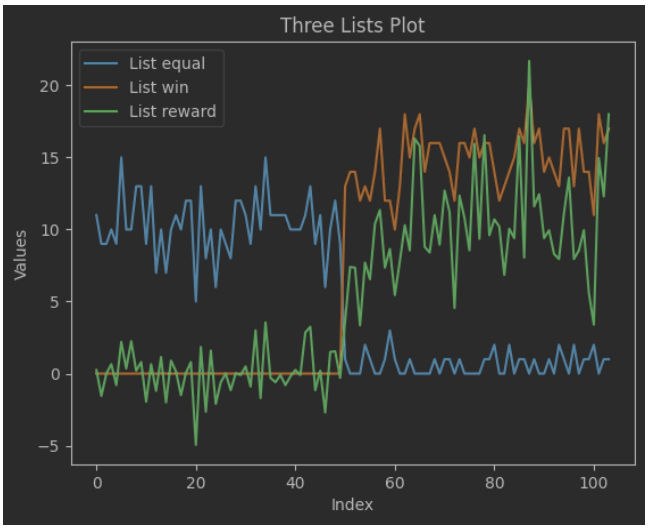
سپس به ازای تعداد مشخصی از episodes هایی که مشخص کردیم عملیات یادگیری را شروع می کنیم .
ابتدا اطلاعات محیط بازی را از سیستم می گیریم و سپس به کمک اطلاعات بدست آمده یک اکشن را اختیار می کنیم .
چون می‌خواهیم مهره سفید را ابتدا آموزش دهیم رفتار مهره سفید را با Sarsa و رفتار مهره سیاه را به صورت رندوم قرار می‌دهیم.

```
        elif agent == 'second_0':
            rand_num = random.randint(0, 17)
            env.step(rand_num)
            observation, reward, termination, truncation, info = env.last()

            done = termination or truncation

        if done:
            games_reward.append(tot_rew)
            break
    # Test the policy every 300 episodes and print the results
    if (ep % 100) == 99:
        with open(f'checkpoint_main/v4_Q_Sarsa_learn_by_random_agent{ep}.pkl', 'wb') as f:
            pickle.dump(Q, f)
    if (ep % 300) == 299:
        print("I am in testing mood")
        with open('checkpoint_main/v4_Q_Sarsa_learn_by_random_agent.pkl', 'wb') as f:
            pickle.dump(Q, f)
        test_rew = run_episodes(env, Q, 10)

        print("Episode:{:5d}  Eps:{:2.4f}  Rew:{:2.4f}".format(ep, eps, test_rew))
        test_rewards.append(test_rew)
        with open('checkpoint_main/v4_test.pkl', 'wb') as f:
            pickle.dump(test_rew, f)

        print("Testing mood is end")

return Q
```

در قسمت به کمک eps_greedy لیست محیطی که دیدم و observation فعلی که داریم بهترین اکشن موجود را انتخاب می‌کنیم یا یک با یک احتمالی این کار را نمی‌کنیم و یک اکشن رندوم انجام می‌دهیم تا بتوانیم محیط را بهتر ببینیم .
اگر محیطی که دیدیم جز محیطی نباشد که قبلا دیده‌ایم یک اکشن را به صورت رندوم انتخاب می‌کنیم همچنین اگر چند اکشن باشند که یک امتیاز داشته باشند از بین آنان یکی را به صورت رندوم انتخاب می‌کنیم.

```python
def eps_greedy(Q, s, eps=0.1):
    '''
    Epsilon greedy policy
    '''
    if np.random.uniform(0, 1) < eps:
        # Choose a random action
        return random.randint(0, 17)
    else:
        # Choose the action of a greedy policy
        return greedy(Q, s)


def greedy(Q, s):
    '''
    Greedy policy
    return the index corresponding to the maximum action-state value
    '''
    if s not in Q.keys():
        Q[s] = [0 for _ in range(18)]

    most_value = np.argmax(Q[s])
    index_max = [_ for _ in range(18) if Q[s][_] == Q[s][most_value]]
    # print(index_max)
    index = random.randint(0, len(index_max) - 1)

    return index_max[index]
```

پس از اتمام آموزش نمودارها بدست آمده از تست را رسم می‌کنیم که در این نمودارها تعداد مساوی انجام شده و تعداد بردها و میزان میانگین امتیاز به ازای هر ۲۰ بازی به ازای هر ۱۰۰ بار آموزش بدست آمده است که هر عدد محور افقی برابر با تست مدل Sarca بعد از ۱۰۰ بازی است .



نکته آنکه برای این سوال فایل‌های pkl مربوط به یادگیری قرار گرفته نشده است (به ازای هر ۱۰۰ بار بازی یک فایل pikl برای ذخیره مدل ساخته می‌شد) زیرا حجم فایل آپلودی را به یک گیگ میرساند .

سوال دوم :

قسمت اول بررسی روش‌های model-based :

کمک گرفته شده از کد :

https://github.com/SparkShen02/MDP-with-Value-Iteration-and-Policy-Iteration/blob/main/valueIteration.py#LL12C1-L30C15

هدف ما آپدیت کردن ارزش هر خانه و state براساس فرمولیشن iteration value هست که برابر است با :

procedure *Value_iteration*(S, A, P, R)
  Inputs
    S is the set of all states
    A is the set of all actions
    P is state transition function specifying $P(s' \mid s, a)$
    R is a reward function $R(s, a)$
  Output
    $\pi[S]$ approximately optimal policy
    $V[S]$ value function
  Local
    real array $V_k[S]$ is a sequence of value functions
    action array $\pi[S]$
  assign $V_0[S]$ arbitrarily
  $k := 0$
  repeat
    $k := k + 1$
    for each state s **do**
      $V_k[s] = \max_a R(s, a) + \gamma * \sum_{s'} P(s' \mid s, a) * V_{k-1}[s']$
  until termination
  for each state s **do**
    $\pi[s] = \arg\max_a R(s, a) + \gamma * \sum_{s'} P(s' \mid s, a) * V_k[s']$
  return π, V_k

Figure 9.16: Value iteration for MDPs, storing V

```python
# https://github.com/SparkShen02/MDP-with-Value-Iteration-and-Policy-Iteration/blob/main/valueIteration.py#LL12C1-L30C15
def valueIteration(env):
    print("During the value iteration:\n")
    U = np.zeros((4, 4))
    nextU = np.zeros((4, 4))
    while True:
        # nextU = np.zeros((4, 4))
        error = 0
        for r in range(NUM_ROW):
            for c in range(NUM_COL):
                nextU[r][c] = max([calculateU(env, U, r, c, action) for action in range(NUM_ACTIONS)])  # Bellman update
                error = max(error, abs(nextU[r][c] - U[r][c]))


        # env.map = nextU
        U = nextU.copy()
        print(U)
        print(error)
        if error < 0.25:
            break
    return U
```

در درون حلقه لوپ بیرونی ابتدا به ازای تمامی خانه‌های درون محیط آزمایش یک آرایه صفر می‌سازیم، سپس در قدم بعد تا زمانی که میزان ارور ما از میزان ماکزیمم ارور در حاصل کسر یک منهای DISCOUNT تقسیم بر DISCOUNT شود انگاه عملیات را متوقف می کنیم.

در درون این حلقه یک حلقه for دیگر داریم که به ازای تمامی خانه‌های سعی می کند به کمک فرمولیشن belaman براساس مشاهدات فعلی و قبلی میزان ارزش خانه جدید را آپدیت کند و درنهایت پس از تمام شدن یک دور ارزش گذاری خانه‌ها، میزان ارزش کل خانه ها را تغییر می‌دهیم .

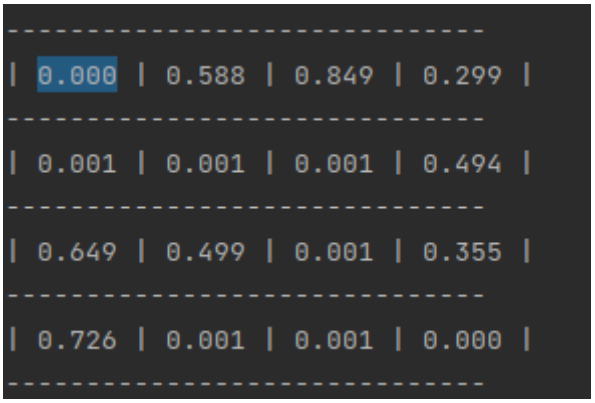
میزان ارزش هر خانه به ازای هر اکشن به شکل زیر محاسبه می‌شود .

```python
# Calculate the utility of a state given an action
def calculateU(env, u, row, c, action):
    states, probs, fail_probs, dones = env.possible_consequences(action, state_now=(row, c))

    list_of_next_idx_we_can_go = np.arange(len(states))
    list_of_reward_each_states_near = []
    for i in range(len(list_of_next_idx_we_can_go)):

        next_idx = list_of_next_idx_we_can_go[i]
        r = REWARD
        done = dones[next_idx]
        if done:
            r += 50
        elif np.random.rand() < fail_probs[next_idx]:
            r -= 10

        list_of_reward_each_states_near.append(probs[i] * (r + DISCOUNT * u[row][c]))

    return sum(list_of_reward_each_states_near)
```

باید در نظر داشت که REWARD به صور پایه برابر با منفی یک است که در صورت قرار گرفتن در خانه‌ی هدف برابر با 50 و در صورت سقوط منفی ۱۰ امتیاز خواهد گرفت و در نهایت حاصل نهایی در مقدار DISCOUNT و احتمالی که این اکشن رخ بدهد ضرب خواهد شد که متناسب با خواست مسئله است . سپس این مقادیر باهم جمع خواهند شد و به عنوان حاصل یک اکشن برگشت خواهند داده شد همانطور که می‌دانیم برای هر اکشن چند احتمال وجود دارد که یکی آن است که اکشن با احتمال ۹۰ درصد اجرا شود و حال آنکه ممکن است اکشن‌های دیگر با مجموعه احتمال ۱۰ درصد نیز رخ بدهند که باید برای حاصل نهایی اکشن جمع تمامی این مقادیر در احتمالاتشان را داشته و حاصل نهایی را برگردانیم .

در نهایت حاصل بدست آمده به شکل زیر خواهد بود :

شکل اولیه تولید شده توسط محیط به صورت رندوم برای احتمال یخ :



حاصل بدست آمده در مراحل بعد برای ارزش هر خونه :

```
During the value iteration:

[[-1.   -1.   -1.25 -1.  ]
 [-1.   -1.25 -1.   -1.25]
 [-1.5  -1.   -1.   45.  ]
 [-1.25 -1.   45.25 46.25]]
46.25000000000001
[[-1.9   -1.9   -2.125 -2.15 ]
 [-1.9   -2.375 -2.4   -2.125]
 [-2.6   -1.9   -2.15  85.5  ]
 [-2.375 -2.15  85.975 87.875]]
41.62499999999999
[[ -2.71    -2.96    -3.1625  -2.935 ]
 [ -2.71    -3.1375  -3.41    -3.1625]
 [ -3.59    -2.71    -2.935  122.2   ]
 [ -3.8875  -3.185  122.6275 125.5875]]
37.71250000000002
[[ -3.439    -3.914    -3.84625  -4.1415 ]
 [ -3.439    -3.82375  -4.069    -3.84625]
 [ -4.731    -3.439    -3.6415  154.98   ]
 [ -4.49875  -4.3665  155.61475 159.52875]]
33.94124999999998
[[ -4.0951    -4.7726    -4.461625  -4.97735 ]
 [ -4.0951    -4.441375  -4.9121    -4.461625]
 [ -5.7579    -4.0951    -4.77735  184.232   ]
 [ -5.048875  -5.17985  185.303275 190.075875]]
30.547124999999994
[[ -4.68559    -5.54534    -5.5154625  -5.729615 ]
 [ -4.93559    -4.9972375  -5.67089    -5.0154625]
 [ -6.18211    -4.68559    -5.299615  211.0588   ]
 [ -6.0439875  -5.661865  212.0229475 217.5682875]]
27.49241250000003
```

```
20.041968712499965
[[ -6.6954656   -7.68704757  -7.96269495  -7.8554504 ]
 [ -6.8594906   -7.19268752  -7.58717093  -7.15714495]
 [ -8.64758237  -6.7632156   -7.1660774  293.45567868]
 [ -8.8312102   -8.21599963 294.54075585 302.66005343]]
18.03777184125005
[[ -7.27591904  -8.16834282  -8.66642545  -8.31990536]
 [ -7.17354154  -7.47341877  -8.07845384  -7.44143045]
 [ -9.03282413  -7.33689404  -7.44946966 309.36011081]
 [ -8.94808918  -8.64439966 310.33668027 318.89404809]]
16.233994657125038
[[ -7.79832714  -8.35150853  -9.29978291  -8.73791483]
 [ -7.45618739  -7.72607689  -8.77060845  -7.94728741]
 [ -9.62954172  -7.60320464  -7.7045227  323.42409973]
 [ -9.80328026  -8.7799597  324.55301224 333.50464328]]
14.610595191412585
[[ -8.01849442  -8.76635768  -9.36980462  -9.11412334]
 [ -7.71056865  -8.2034692   -9.14354761  -8.40255867]
 [ -9.66658755  -7.84288417  -8.18407043 336.08168976]
 [ -9.82295223  -8.90196373 337.34771102 346.65417895]]
13.149535672271327
[[ -8.21664498  -9.38972191  -9.93282415  -9.45271101]
 [ -7.93951178  -8.63312228  -9.47919285  -8.8123028 ]
 [-10.19992879  -8.05859575  -8.36566338 347.47352078]
 [-10.59065701  -9.26176735 348.86293992 358.48876105]]
11.834582105044092
[[ -8.39498048  -9.70074972 -10.18954174  -9.75743991]
 [ -8.3955606   -9.01981006 -10.03127356  -8.93107252]
 [-10.42993591  -8.50273618  -8.77909704 357.7261687 ]
 [-10.53159131  -9.33559062 359.22664592 369.13988495]]
10.65112389453975
[[ -8.55548243  -9.73067475 -10.67058756 -10.28169592]
 [ -8.55600454  -9.11782905 -10.27814621  -9.28796527]
 [-10.88694232  -8.65246256  -8.90118734 367.20355183]
 [-10.47843218  -9.40203156 368.55398133 378.72589645]]
9.586011505085764
```

```
9.586011505085764
[[ -8.69993419  -9.75760727 -10.85352881 -10.50352633]
 [ -8.70040409  -9.45604615 -10.50033159  -9.35916874]
 [-11.04824809  -8.78721631  -9.26106861 375.48319665]
 [-10.68058896  -9.4618284  376.9485832  387.35330681]]
8.627410354577137
[[ -8.82994077 -10.03184655 -11.26817593 -10.70317369]
 [ -8.83036368  -9.51044153 -10.70029843  -9.92325187]
 [-11.44342328  -8.90849467  -9.33496175 382.93487699]
 [-10.61253007  -9.51564556 384.50372488 395.11797613]]
7.764669319119491
[[ -8.94694669 -10.02866189 -11.64135833 -10.88285632]
 [ -8.94732731  -9.80939738 -10.88026858 -10.18092668]
 [-11.54908095  -9.26764521  -9.40146557 389.89138929]
 [-11.05127706  -9.81408101 391.30335239 402.10617852]]
6.98820238720748
[[ -9.05225202 -10.5257957  -11.7272225  -10.79457069]
 [ -9.05259458 -10.07845764 -11.04224173 -10.66283401]
 [-11.64417286  -9.34088069  -9.46131901 395.90225036]
 [-11.44614935 -10.0826729  397.42301715 408.39556066]]
6.289382148486823
[[ -9.14702682 -10.72321613 -11.80450025 -10.96511362]
 [ -9.14733512 -10.07061188 -10.93801755 -10.84655061]
 [-12.22975557  -9.40679262  -9.51518711 401.31202532]
 [-11.55153442 -10.32440561 402.93071544 413.8060046 ]]
5.50769828479406
[[ -9.23232414 -10.90089452 -11.87405023 -11.11860226]
 [ -9.23260161 -10.31355069 -11.0942158  -10.76189555]
 [-12.50678001  -9.46611336  -9.5636684  406.43082279]
 [-11.89638098 -10.29196505 407.88764389 418.92540414]]
5.11939954027423
[[ -9.30909173 -11.06080507 -11.9366452  -11.25674203]
 [ -9.30934145 -10.28219562 -10.98479422 -10.93570599]
 [-12.75610201  -9.51950202  -9.60730156 410.78774051]
 [-11.70674288 -10.51276855 412.3488795  423.53286372]]
```

```
 [-13.42089203  -9.95826171 -10.07954864 440.91643464]
 [-13.07636195 -11.91968322 444.23324035 456.31792278]]
1.4441845842817855
[[ -9.8719733  -11.92898411 -13.103283   -24.19789513]
 [-10.1713747  -11.04469524 -11.90693719 -10.49631848]
 [-13.32880283  -9.96243554 -10.07159378 441.57479118]
 [-13.26872575 -11.7277149  445.05991631 457.1861305 ]]
1.299766125853612
[[ -9.88477597 -11.9860857  -13.0429547  -23.27810562]
 [-10.15423723 -10.94022571 -11.71624347 -10.44668663]
 [-13.24592255  -9.96619198 -10.0644344  442.66731206]
 [-13.44185318 -11.55494341 445.80392468 457.96751745]]
1.0925208823875892
[[-10.14629837 -12.03747713 -12.98865923 -22.20029506]
 [-10.13881351 -11.09620314 -11.79461912 -10.65201797]
 [-13.17133029 -10.21957279 -10.30799096 443.40058085]
 [-13.34766786 -11.39944907 446.47353221 458.6707657 ]]
1.0778105619414227
[[-10.13166854 -12.08372941 -12.9397933  -21.23026555]
 [-10.12493216 -10.98658283 -11.86515721 -10.83681617]
 [-13.10419726 -10.19761551 -10.27719186 444.06052277]
 [-13.76290107 -11.50950416 447.07617899 459.30368913]]
0.9700295057472808
[[-10.11850168 -12.12535647 -13.14581397 -20.357239  ]
 [-10.11243894 -11.13792455 -12.17864149 -10.75313456]
 [-13.04377754 -10.17785396 -10.24947268 444.65447049]
 [-13.63661097 -11.35855374 447.61856109 459.87332022]]
0.8730265551725509
[[-10.10665152 -12.16282082 -13.33123258 -19.5715151 ]
 [-10.35119505 -11.27413209 -12.21077734 -10.9278211 ]
 [-12.98939978 -10.16006856 -10.22452541 444.93902344]
 [-13.52294987 -11.22269837 448.10670498 460.3859882 ]]
0.7857238996552951
```

```
0.30530593118109906
[[-10.55107863 -12.4058421  -12.39313144 -14.97297804]
 [-10.29245873 -10.97310499 -11.761709   -10.86243509]
 [-12.74146706 -10.50494658 -10.48972276 448.58091367]
 [-14.29255735 -11.04091108 451.12133747 463.55207421]]
0.35638251633657525
[[-10.49597076 -12.16525789 -12.40381829 -14.72568024]
 [-10.26321285 -10.87579449 -11.8355381  -10.77619158]
 [-12.96732035 -10.45445192 -10.69075048 448.9728223 ]
 [-13.86330162 -11.18681997 451.25920372 463.44686679]]
0.4292557352970814
[[-10.69637369 -12.1987321  -12.41343646 -14.75311221]
 [-10.48689157 -10.78821504 -11.90198429 -10.94857242]
 [-13.17058832 -10.40900673 -10.62167543 449.32554007]
 [-13.97697146 -11.31813797 451.38328335 463.60218011]]
0.35271776995352866
[[-10.62673632 -11.97885889 -12.92209282 -14.77780099]
 [-10.68820241 -10.70939353 -11.96178586 -11.10371518]
 [-12.85352949 -10.36810606 -10.55950789 449.39298606]
 [-13.57927431 -11.43632417 451.49495502 463.7419621 ]]
0.5086563535562796
[[-10.56406269 -12.030973   -12.87988354 -14.30002089]
 [-10.61938217 -10.63845418 -11.76560728 -10.99334366]
 [-13.06817654 -10.33129545 -10.5035571  449.45368746]
 [-13.22134688 -11.54269176 451.59545952 463.86776589]]
0.4777800993031285
[[-10.50765642 -12.0778757  -12.84189518 -14.1200188 ]
 [-10.55744395 -10.57460876 -11.83904655 -10.8940093 ]
 [-13.01135888 -10.29816591 -10.70320139 449.50831871]
 [-13.14921219 -11.63842258 451.68591356 463.9809893 ]]
0.19964429003305817
```

در نتیجه براساس نتایج آخر بدست آمده بهترین policy ممکن برای هر خانه را بدست آوردم :

```
The optimal policy is:

| down  | <-    | down  | down  |
| up    | down  | <-    | down  |
| ->    | up    | down  | down  |
| ->    | ->    | ->    | G     |
```

در قسمت دوم این تمرین به بررسی model-free می‌پردازیم :

https://github.com/PacktPublishing/Reinforcement-Learning-Algorithms-with-Python/blob/master/Chapter04/SARSA%20Q_learning%20Taxi-v2.py

از همین رو دو مدل Q-learning , Sarsa را پیاده سازی کردم .

مدل Sarsa را در مسئله شماره یک قبلا پیاده سازی کردیم، Q-learning شباهت بسیار زیادی به این الگوریتم دارد و صرفا در آپدیت کردن میزان ارزش هر خانه یا state با هم تفاوت کوچکی در محاسبه دارند که در پایین می‌بینیم .

Q-learning :

```
# Q-learning update the state-action value (get the max Q value for the next state)
Q[state][action] = Q[state][action] + lr * (rew + gamma * np.max(Q[next_state]) - Q[state][action])
```

Sarsa :

```
# SARSA update
Q[state][action] = Q[state][action] + lr * (rew + gamma * Q[next_state][next_action] - Q[state][action])
```

در سیاست Sarsa ما برای آپدیت کردن میزان ارزش هر state متناسب با اکشن آن یک مرحله جلوتر را نگاه می‌کنیم و متناسب با بهترین اکشن انجام گرفته شده در مرحله بعد براساس سیاست epsilon gready میزان ارزش خانه را تغییر می‌دهیم. حال آنکه در Q-learning این میزان ارزش را براساس ماکزیم ارزش در state بعد بدست می‌آوریم .

در اینجا gamma همان discont factor می‌باشد .

پس انجام اینکار به ازای هر epoch بیست‌بار آزمایش انجام می‌دهیم و متوسط مجموع پاداش را ذخیره می‌کنیم .

که نمودار زیر حاصل می‌شود .