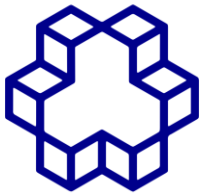
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

مینی پروژه چهارم

Github link:

https://github.com/FatemehShokrollahiMoghadam

google drive link:

https://drive.google.com/drive/folders/1oFh5an35Q6h42DEsNA36EbEkoOJjOI_l?usp=sharing

نگارنده:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

تابستان ۱۴۰۳

فهرست

پیشگفتار

در صفحه اول گزارش لینگ گیت هاب و لینک گوگل درایو مربوط به نوت بوک های هر سوال آورده شده است.

در اینجا نیز لینک گوگل کولب نوت بوک آورده می‌شود:

لینک نوت بوک سوال اول:

https://colab.research.google.com/drive/1BTQARZh2kG_GfuaEMkVxfA-gHcq6xL_B?usp=sharing

سوال اول

آ)

برای مسئله wumpus world یکبار با روش Q-Learning و یکبار با روش Deep Q-Learning عاملی را طراحی کرده و آموزش دهید.

محیط بازی در gym قابل استفاده نبود. ابتدا محیط را ساخته و سپس به روند طراحی عوامل با روش های Q-Learning و Deep Q-Learning می پردازیم.

۱-آ طراحی محیط

برای طراحی محیط wumpus world کلاسی را تحت عنوان WumpusWorldEnv می سازیم. در این کلاس از متد های update_gridi nit, reset, step, move, shoot, get_reward, is_done, استفاده شده است.

متد init محیط را با شبکه ای به ابعاد ۴*۴ مقداردهی اولیه می کند و متد reset را فراخوانی می کند.

```
class WumpusWorldEnv:
    def __init__(self):
        self.size = 4
        self.reset()
```

متد reset شبکه را مقدار دهی اولیه می کند، طلا را در موقعیت (۳، ۳)، wumpus ها وچاله ها را در موقعیت های از پیش تعریف شده قرار می دهد، عامل را در موقعیت (۰، ۰) قرار می دهد و موقعیت اولیه عامل را برمی گرداند.

```
def reset(self):
    self.grid = np.zeros((self.size, self.size), dtype=int)
    self.grid[3, 3] = 2  # Gold
    self.wumpus_positions = [ (1, 1), (2, 3)]
    for pos in self.wumpus_positions:
        self.grid[pos] = 3  # Wumpus
    self.pit_positions = [ (0, 2), (3, 2)]  # Define pit positions
    for pos in self.pit_positions:
        self.grid[pos] = -1  # Pit
    self.agent_pos = [0, 0]
    self.grid[self.agent_pos[0], self.agent_pos[1]] = 1
    self.arrow = True
    return tuple(self.agent_pos)
```

متد step یک action را به عنوان ورودی می گیرد. اگر عمل یک حرکت باشد (۰=چپ، ۱=راست، ۲=بالا، ۳=پایین)، متد move را برای به‌روزرسانی موقعیت عامل فراخوانی می‌کند. اگر عمل شلیک باشد (۴=چپ، ۵=راست، ۶=بالا، ۷=پایین)، متد shoot را برای کنترل تیرهای شلیک شده فراخوانی می کند. سپس پاداش را محاسبه می کند و بررسی می کند که آیا بازی انجام شده است یا خیر، و بر این اساس شبکه را به روز می کند.

```python
def step(self, action):
    reward = 0

    # Move actions: 0=left, 1=right, 2=up, 3=down
    if action < 4:
        self.move(action)
    else:
        reward = self.shoot(action - 4)

    reward += self.get_reward()
    done = self.is_done()
    self.update_grid()

    return tuple(self.agent_pos), reward, done
```

متدshoot (امتیازی) به عامل اجازه می دهد تا یک تیر را در جهت معینی شلیک کند و بررسی کند که آیا تیر به wumpus برخورد می کند یا خیر. اگر به wumpus برخورد کند، شبکه را به روز می کند و به عامل پاداش می دهد.

```python
def shoot(self, direction):
    reward = 0
    if self.arrow:
        self.arrow = False
        if direction == 0:  # left
            for i in range(self.agent_pos[1], -1, -1):
                if self.grid[self.agent_pos[0], i] == 3:
                    self.grid[self.agent_pos[0], i] = 0
                    reward = 50
                    self.wumpus_positions.remove((self.agent_pos[0], i))
                    break
        elif direction == 1:  # right
            for i in range(self.agent_pos[1], self.size):
                if self.grid[self.agent_pos[0], i] == 3:
                    self.grid[self.agent_pos[0], i] = 0
                    reward = 50
                    self.wumpus_positions.remove((self.agent_pos[0], i))
                    break
        elif direction == 2:  # up
            for i in range(self.agent_pos[0], -1, -1):
                if self.grid[i, self.agent_pos[1]] == 3:
                    self.grid[i, self.agent_pos[1]] = 0
                    reward = 50
                    self.wumpus_positions.remove((i, self.agent_pos[1]))
                    break
        elif direction == 3:  # down
            for i in range(self.agent_pos[0], self.size):
                if self.grid[i, self.agent_pos[1]] == 3:
                    self.grid[i, self.agent_pos[1]] = 0
                    reward = 50
                    self.wumpus_positions.remove((i, self.agent_pos[1]))
                    break
    return reward
```

متدget_reward بر اساس موقعیت فعلی عامل، پاداش را برمی‌گرداند (به عنوان مثال، ۱۰۰ برای یافتن طلا، ۱۰۰۰- برای مواجهه با wumpus یا چاله و در غیر این صورت۱-).

```
def get_reward(self):
    if self.grid[self.agent_pos[0], self.agent_pos[1]] == 2:  # Gold
        return 100
    elif self.grid[self.agent_pos[0], self.agent_pos[1]] == 3:  # Wumpus
        return -1000
    elif self.grid[self.agent_pos[0], self.agent_pos[1]] == 4:  # Pit
        return -1000
    return -1
```

متد is_done بررسی می‌کند که بازی بر اساس موقعیت عامل به پایان رسیده است یا خیر.

```
def is_done(self):
    return self.grid[self.agent_pos[0], self.agent_pos[1]] in [2, 3, 4]
```

متد Update_grid شبکه را با موقعیت های عامل و wumpus و چاله به روز می کند.

```
def update_grid(self):
    self.grid = np.zeros((self.size, self.size), dtype=int)
    self.grid[3, 3] = 2  # Gold
    for pos in self.wumpus_positions:
        self.grid[pos] = 3  # Wumpus
    for pos in self.pit_positions:
        self.grid[pos] = 4  # Pit
    self.grid[self.agent_pos[0], self.agent_pos[1]] = 1  # Agent position
```

۲-آ طراحی عامل Q_Learning

حال عامل را با استفاده از Q Learning طراحی می کنیم. برای این منظور کلاس QLearningAgent را تشکیل می دهیم.

کلاس QLearningAgent متد های init, get_q_value, update_q_value, choose_action,train, plot_cumulative_reward, play است.

در متد init ضریب تخفیف و نرخ یادگیری را مطابق خواسته سوال به ترتیب مقادیر 0.9 و 0.1 در نظر می گیریم. همچنین نرخ اکتشاف را ۱ در نظر می گیریم به این معنی که عامل با احتمال بالایی برای انجام عمل تصادفی شروع می کند. نرخ کاهش آن را 0.995 در نظر می گیریم به این معنی که پس از هر قسمت، نرخ اکتشاف در ۰,۹۸ ضرب می شود و به آرامی احتمال انجام عمل تصادفی کاهش می دهد. در این متد یک دیکشنری برای ذخیره جدول Q و یک لیست برای مقدار پاداش تجمعی تعریف می شود.

همچنین برای یافتن تعداد اپیزود لازم برای یادگیری سیاست بهینه با استفاده از self.episodes_to_consistency بررسی می شود.

```
class QLearningAgent:
    def __init__(self, env, learning_rate=0.1, discount_factor=0.9, exploration_rate=1.0, exploration_decay=0.995):
        self.env = env
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.exploration_decay = exploration_decay
        self.q_table = {}
        self.cumulative_rewards = []  # To store cumulative rewards
        self.consecutive_successes_needed = 5  # Number of consecutive successful episodes to achieve consistency
        self.consecutive_successes = 0
        self.episodes_to_consistency = None
```

متدget_q_value مقدار Q را برای یک جفت حالت-عمل معین از جدول Q بازیابی می کند.

```
def get_q_value(self, state, action):
    return self.q_table.get((state, action), 0.0)
```

در متد update_q_value معادله بلمن پیاده سازی می شود. بهترین مقدار را max ارزش محاسبه شده در انجام بهترین عمل در هر حالت در نظر می گیرد. مقدار Value در هر حالت با انجام یک عمل باتوجه به مقدار گذشته و مقدار جدید ناشی از عمل جدید آپدیت می شود.

```
def update_q_value(self, state, action, reward, next_state):
    best_next_action = max(range(8), key=lambda x: self.get_q_value(next_state, x))
    td_target = reward + self.discount_factor * self.get_q_value(next_state, best_next_action)
    td_error = td_target - self.get_q_value(state, action)
    new_q_value = self.get_q_value(state, action) + self.learning_rate * td_error
    self.q_table[(state, action)] = new_q_value
```

متد choose_action بر اساس نرخ اکتشاف، عملی را برای عامل انتخاب می کند. یا یک عمل تصادفی (اکتشاف) یا عملی را با بیشترین مقدار Q برای وضعیت فعلی انتخاب می کند.

```
def choose_action(self, state):
    if random.random() < self.exploration_rate:
        return random.randint(0, 7)
    else:
        return max(range(8), key=lambda x: self.get_q_value(state, x))
```

متد train قسمت هایی از آموزش را اجرا می کند که در آن عامل با محیط تعامل می کند، عمل ها را انتخاب می کند، مقادیر Q را به روز می کند و پاداش تجمعی را محاسبه می کند. همچنین سرعت اکتشاف را در طول زمان کاهش می دهد.

```
def train(self, episodes, max_steps_per_episode):
    for episode in range(episodes):
        state = self.env.reset()
        total_reward = 0

        for step in range(max_steps_per_episode):
            action = self.choose_action(state)
            next_state, reward, done = self.env.step(action)
            self.update_q_value(state, action, reward, next_state)
            state = next_state
            total_reward += reward

            if done:
                break

        self.exploration_rate *= self.exploration_decay
        self.cumulative_rewards.append(total_reward)  # Store cumulative reward

        # Check for consecutive successes
        if reward == 100:  # Reached gold
            self.consecutive_successes += 1
        else:
            self.consecutive_successes = 0

        if self.consecutive_successes == self.consecutive_successes_needed and self.episodes_to_consistency is None:
            self.episodes_to_consistency = episode + 1

        print(f"Episode {episode + 1}: Total Reward: {total_reward}, Exploration Rate: {self.exploration_rate}")
```

متد play یک ایزود را اجرا می کند. در حالت اولیه مجموع پاداش بدست آمده صفر است. ابتدا با ریست کردن محیط به حالت شروع می رویم. در هر حالت بهترین عمل از نظر مقدار Q-value انتخاب می شود و در محیط اجرا شده و حالت جدید و پاداش آن دریافت می شود و به مجموع پاداش ها اضافه می شود. اگر به آخرین اپیزود رسید، بازی تمام شود و الا به حالت جدید برود و مراحل را تکرار کند.

```
def play(self, episodes, max_steps_per_episode):
    for episode in range(episodes):
        state = self.env.reset()
        total_reward = 0

        for step in range(max_steps_per_episode):
            action = self.choose_action(state)
            next_state, reward, done = self.env.step(action)
            state = next_state
            total_reward += reward

            if done:
                break

        print(f"Episode {episode + 1}: Total Reward: {total_reward}")
```

متد culmutive_reward پاداش تجمعی به‌دست‌آمده در هر اپیزود آموزش را ترسیم می‌کند و پیشرفت یادگیری عامل را در طول قسمت‌ها نشان می‌دهد.

```
def plot_cumulative_rewards(self):
    plt.plot(self.cumulative_rewards)
    plt.title('Cumulative Reward during Training')
    plt.xlabel('Episode')
    plt.ylabel('Cumulative Reward')
    plt.grid(True)
    plt.show()
```

در نهایت محیط WumpusWorldEnv و عامل Q _Learning فراخوانی می شود.عامل را برای تعداد مشخصی اپیزود (۱۰۰۰)و تعداد (۵۰)در هر اپیزود آموزش می دهیم. و در آخر پاداش تجمعی به دست آمده در طول آموزش را ترسیم می کند. سپس بازی را با عامل آموزش دیده برای ۱۰ اپیزود انجام می دهیم.

```python
if __name__ == "__main__":
    env = WumpusWorldEnv()
    agent = QLearningAgent(env)
    # Training the agent
    agent.train(episodes=1000, max_steps_per_episode=50)
    # Plot cumulative rewards
    agent.plot_cumulative_rewards()
    # Playing the game with the trained agent
    agent.play(episodes=10, max_steps_per_episode=50)
```

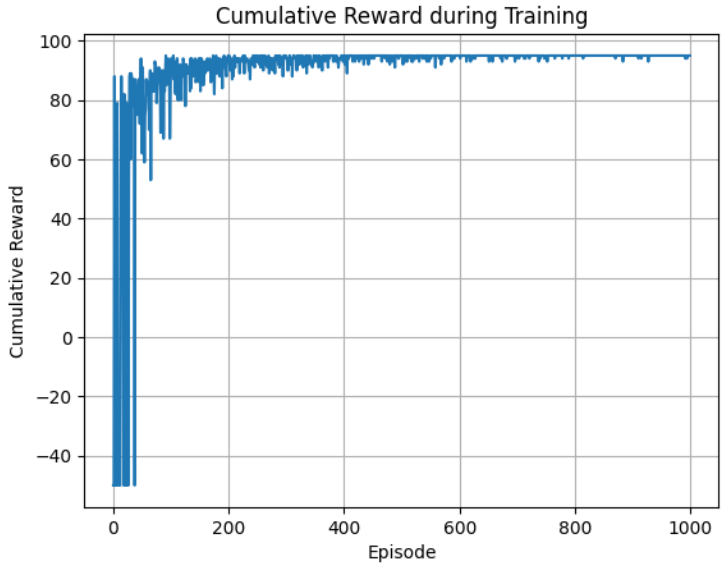
نتایج به‌صورت زیر قابل نمایش است:

مقادیر پاداش و نرخ اکتشاف برای اپیزود های اولیه و نهایی به صورت زیر است:

```
Episode 1: Total Reward: -1015, Exploration Rate: 0.98
Episode 2: Total Reward: -984, Exploration Rate: 0.9603999999999999
Episode 3: Total Reward: -1027, Exploration Rate: 0.9411919999999999
Episode 4: Total Reward: -1018, Exploration Rate: 0.9223681599999999
Episode 5: Total Reward: -999, Exploration Rate: 0.9039207967999998
Episode 6: Total Reward: 0, Exploration Rate: 0.8858423808639998
Episode 7: Total Reward: -1003, Exploration Rate: 0.8681255332467198
Episode 8: Total Reward: -1034, Exploration Rate: 0.8507630225817854
Episode 9: Total Reward: -1011, Exploration Rate: 0.8337477621301497
Episode 10: Total Reward: -1021, Exploration Rate: 0.8170728068875467
Episode 11: Total Reward: -964, Exploration Rate: 0.8007313507497957
Episode 12: Total Reward: -1016, Exploration Rate: 0.7847167237347998
Episode 13: Total Reward: -958, Exploration Rate: 0.7690223892601038
Episode 14: Total Reward: -966, Exploration Rate: 0.7536419414749017
Episode 15: Total Reward: -970, Exploration Rate: 0.7385691026454037
Episode 16: Total Reward: -50, Exploration Rate: 0.7237977205924956
Episode 17: Total Reward: -995, Exploration Rate: 0.7093217661806457
Episode 18: Total Reward: 0, Exploration Rate: 0.695353308570327
Episode 19: Total Reward: -1028, Exploration Rate: 0.6812326242398921
Episode 20: Total Reward: 0, Exploration Rate: 0.6676079717550942
Episode 21: Total Reward: 0, Exploration Rate: 0.6542558123199923
Episode 22: Total Reward: 0, Exploration Rate: 0.6411706960735924
Episode 23: Total Reward: 0, Exploration Rate: 0.6283472821521205
Episode 24: Total Reward: -975, Exploration Rate: 0.6157803365090782
Episode 25: Total Reward: 0, Exploration Rate: 0.6034647297788965
Episode 26: Total Reward: -966, Exploration Rate: 0.5913954351833186
Episode 27: Total Reward: 0, Exploration Rate: 0.5795675264796523
Episode 28: Total Reward: -1007, Exploration Rate: 0.5679761759500592
Episode 29: Total Reward: -978, Exploration Rate: 0.5566166524310581
Episode 30: Total Reward: 0, Exploration Rate: 0.5454843193824369
Episode 31: Total Reward: 0, Exploration Rate: 0.5345746329947881
Episode 32: Total Reward: 0, Exploration Rate: 0.5238831403348924
Episode 33: Total Reward: -50, Exploration Rate: 0.5134054775281945
Episode 34: Total Reward: -50, Exploration Rate: 0.5031373679776306
Episode 35: Total Reward: -992, Exploration Rate: 0.493074620618078
Episode 36: Total Reward: 0, Exploration Rate: 0.48321312820571644
Episode 37: Total Reward: -969, Exploration Rate: 0.4735488656416021
Episode 38: Total Reward: 0, Exploration Rate: 0.46407788832877006
Episode 39: Total Reward: -50, Exploration Rate: 0.45479633056219465
Episode 40: Total Reward: -50, Exploration Rate: 0.44570040395095073
Episode 41: Total Reward: 0, Exploration Rate: 0.4367863958719317
Episode 42: Total Reward: -50, Exploration Rate: 0.42805066795449304
Episode 43: Total Reward: 0, Exploration Rate: 0.41948965459540316
Episode 44: Total Reward: -50, Exploration Rate: 0.4110998615034951
Episode 45: Total Reward: 0, Exploration Rate: 0.4028778642734252
Episode 46: Total Reward: -50, Exploration Rate: 0.39482030698795667
Episode 47: Total Reward: 0, Exploration Rate: 0.38692390084819756
Episode 48: Total Reward: -50, Exploration Rate: 0.3791854228312336
Episode 49: Total Reward: -1007, Exploration Rate: 0.3716017143460887
Episode 50: Total Reward: -1029, Exploration Rate: 0.3641696800871167
Episode 51: Total Reward: -50, Exploration Rate: 0.35688628648537435
Episode 52: Total Reward: -50, Exploration Rate: 0.34974856075566685
```

```
Episode 949: Total Reward: 95, Exploration Rate: 4.715696346278345e-09
Episode 950: Total Reward: 95, Exploration Rate: 4.621382419352778e-09
Episode 951: Total Reward: 95, Exploration Rate: 4.528954770965722e-09
Episode 952: Total Reward: 95, Exploration Rate: 4.438375675546407e-09
Episode 953: Total Reward: 95, Exploration Rate: 4.349608162035479e-09
Episode 954: Total Reward: 95, Exploration Rate: 4.262615998794769e-09
Episode 955: Total Reward: 95, Exploration Rate: 4.177363678818873e-09
Episode 956: Total Reward: 95, Exploration Rate: 4.093816405242496e-09
Episode 957: Total Reward: 95, Exploration Rate: 4.011940077137646e-09
Episode 958: Total Reward: 95, Exploration Rate: 3.931701275594893e-09
Episode 959: Total Reward: 95, Exploration Rate: 3.853067250082995e-09
Episode 960: Total Reward: 95, Exploration Rate: 3.776005905081335e-09
Episode 961: Total Reward: 95, Exploration Rate: 3.7004857869797088e-09
Episode 962: Total Reward: 95, Exploration Rate: 3.6264760712401147e-09
Episode 963: Total Reward: 95, Exploration Rate: 3.553965498153123e-09
Episode 964: Total Reward: 95, Exploration Rate: 3.482867618819006e-09
Episode 965: Total Reward: 95, Exploration Rate: 3.4132102664426256e-09
Episode 966: Total Reward: 95, Exploration Rate: 3.344946061113773e-09
Episode 967: Total Reward: 95, Exploration Rate: 3.2780471398914974e-09
Episode 968: Total Reward: 95, Exploration Rate: 3.2124861970936674e-09
Episode 969: Total Reward: 95, Exploration Rate: 3.148236473151794e-09
Episode 970: Total Reward: 95, Exploration Rate: 3.085271743688758e-09
Episode 971: Total Reward: 95, Exploration Rate: 3.0235663088149827e-09
Episode 972: Total Reward: 95, Exploration Rate: 2.9630949826386836-09
Episode 973: Total Reward: 95, Exploration Rate: 2.9038330829859094e-09
Episode 974: Total Reward: 95, Exploration Rate: 2.845756421326191e-09
Episode 975: Total Reward: 95, Exploration Rate: 2.788841292899667e-09
Episode 976: Total Reward: 95, Exploration Rate: 2.7330644670416737e-09
Episode 977: Total Reward: 95, Exploration Rate: 2.67840317770084e-09
Episode 978: Total Reward: 95, Exploration Rate: 2.6248351141468232e-09
Episode 979: Total Reward: 95, Exploration Rate: 2.572338411863887e-09
Episode 980: Total Reward: 95, Exploration Rate: 2.5208916436266609e-09
Episode 981: Total Reward: 95, Exploration Rate: 2.4704738107540767e-09
Episode 982: Total Reward: 95, Exploration Rate: 2.421064334538995e-09
Episode 983: Total Reward: 95, Exploration Rate: 2.3726430478482153e-09
Episode 984: Total Reward: 95, Exploration Rate: 2.325190186891251e-09
Episode 985: Total Reward: 95, Exploration Rate: 2.278686383153426e-09
Episode 986: Total Reward: 95, Exploration Rate: 2.2331126554903575e-09
Episode 987: Total Reward: 95, Exploration Rate: 2.1884504023805505e-09
Episode 988: Total Reward: 95, Exploration Rate: 2.1446813943329394e-09
Episode 989: Total Reward: 95, Exploration Rate: 2.1017877664462805e-09
Episode 990: Total Reward: 95, Exploration Rate: 2.059752011117355e-09
Episode 991: Total Reward: 95, Exploration Rate: 2.018556970895008e-09
Episode 992: Total Reward: 95, Exploration Rate: 1.9781858314771076e-09
Episode 993: Total Reward: 95, Exploration Rate: 1.9386221148475656e-09
Episode 994: Total Reward: 95, Exploration Rate: 1.899849672550614e-09
Episode 995: Total Reward: 95, Exploration Rate: 1.8618526790996018e-09
Episode 996: Total Reward: 95, Exploration Rate: 1.8246156255176098e-09
Episode 997: Total Reward: 95, Exploration Rate: 1.7881233130072576e-09
Episode 998: Total Reward: 95, Exploration Rate: 1.7523608467471124e-09
Episode 999: Total Reward: 95, Exploration Rate: 1.7173136298121702e-09
Episode 1000: Total Reward: 95, Exploration Rate: 1.6829673572159268e-09
```

پاداش تجمعی در طی روند آموزش به صورت زیر رسم شد:



همانطور که از شکل بالا پیداست، در اپیزود های اولیه مقدار پاداش بسیار کم بوده و در طی آموزش و پس از حدود ۱۵۰ اپیزود بهبود یافته و در نهایت به مقدار 95 همگرا شده است. بنابراین عامل در اثر تعامل با محیط بازی را آموخته است.

نتایج انجام بازی توسط عامل آموزش دیده شده در ۱۰ اپیزود به صورت زیر است:

```
Episode 1: Total Reward: 95
Episode 2: Total Reward: 95
Episode 3: Total Reward: 95
Episode 4: Total Reward: 95
Episode 5: Total Reward: 95
Episode 6: Total Reward: 95
Episode 7: Total Reward: 95
Episode 8: Total Reward: 95
Episode 9: Total Reward: 95
Episode 10: Total Reward: 95
```

پیداست عامل توانسته است بازی را با پاداش ۹۵ انجام دهد.

۳-آ طراحی عامل Deep Q_Learning

ابتدا کلاس Replay Memory را تعریف می کنیم. این کلاس یک بافر حافظه را مدیریت می کند که در آن تجربیات (وضعیت، عمل، پاداش، حالت بعدی، انجام شدن بازی) ذخیره می شود.

متد Push یک تجربه جدید به حافظه اضافه می کند.

متد sample یک دسته تصادفی از تجربیات را از حافظه بازیابی می کند.

متد len ابعاد فعلی حافظه را برمی گرداند.

```
class ReplayMemory:
    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = deque(maxlen=capacity)

    def push(self, experience):
        self.memory.append(experience)

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

حال کلاس QNetwork را تعریف می کنیم. این کلاس یک شبکه عصبی را تعریف می کند که برای تقریب مقادیر Q برای عامل DQN استفاده می شود. شبکه از ۳ لایه متصل (لایه های خطی) با تابع فعالساز ReLU تشکیل شده است. متد forward مسیر عبور از طریق شبکه را تعریف می کند.

```
class DQNAgent:
    def __init__(self, env, learning_rate=0.1, discount_factor=0.9, exploration_rate=1.0, exploration_decay=0.995,
                 target_update=100, replay_memory_capacity=10000, batch_size=256):
        self.env = env
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.exploration_decay = exploration_decay
        self.target_update = target_update
        self.replay_memory = ReplayMemory(replay_memory_capacity)
        self.batch_size = batch_size

        self.input_size = len(env.reset())
        self.output_size = 8
        self.policy_network = QNetwork(self.input_size, self.output_size)
        self.target_network = QNetwork(self.input_size, self.output_size)
        self.optimizer = optim.Adam(self.policy_network.parameters(), lr=self.learning_rate)

        self.cumulative_rewards = []
        self.steps_done = 0

        # Initialize target network with the same weights as the policy network
        self.update_target_network()
        # Consistency tracking
        self.consecutive_successes_needed = 5
        self.consecutive_successes = 0
        self.episodes_to_consistency = None
```

سپس کلاس DQNAgent را تعریف می کنیم. این کلاس شامل متدهای init, get_q_value, update_target_network, choose_action, push_to_memory, sample_from_memory, learn_from_memory,train, get_cumulative_reward, play است.

متد init عامل DQN را با پارامترها و تنظیمات مختلف مقداردهی می کند. مانند عامل QLearning ضریب تخفیف را 0.9 و نرخ یادگیری را 0.1 می گذاریم. نرخ اکتشاف ۱ و ضریب کاهش آن 0.995 در نظر گرفته شده است. همچنین ظرفیت حافظه ۱۰۰۰۰ و بروزرسانی هدف هر ۱۰۰ تا و اندازه batch را ۲۵۶ در نظر گرفته ایم.

سیاست و هدف شبکه Q را نگه می دارد و روش‌هایی را برای به‌روزرسانی شبکه هدف، دریافت مقادیر Q، انتخاب عمل، ذخیره تجربیات، نمونه‌برداری از حافظه، یادگیری از تجربیات، آموزش عامل، و اجرای اپیزودها پیاده‌سازی می‌کند. از بهینه ساز AdamW استفاده شده است.

ابعاد خروجی ۸ نشان دهنده تعداد عمل های ممکن در محیط است. لایه خروجی شبکه عصبی دارای این تعداد نورون خواهد بود که هر کدام مربوط به یک مقدار Q برای هر عمل است. شبکه Policy یک شبکه عصبی است که برای پیش بینی مقادیر Q برای هرعملی که در یک حالت داده می شود استفاده می شود. شبکه هدف برای ارائه مقادیر Q پایدار در طی آموزش استفاده می شود. این شبکه برای بهبود پایداری در آموزش، کمتر به روز می شود.

```python
class DQNAgent:
    def __init__(self, env, learning_rate=0.1, discount_factor=0.9, exploration_rate=1.0, exploration_decay=0.995, target_update=100, replay_memory_capacity=10000, batch_size=256):
        self.env = env
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.exploration_decay = exploration_decay
        self.target_update = target_update
        self.replay_memory = ReplayMemory(replay_memory_capacity)
        self.batch_size = batch_size

        self.input_size = len(env.reset())
        self.output_size = 8
        self.policy_network = QNetwork(self.input_size, self.output_size)
        self.target_network = QNetwork(self.input_size, self.output_size)
        self.optimizer = optim.Adam(self.policy_network.parameters(), lr=self.learning_rate)

        self.cumulative_rewards = []
        self.steps_done = 0

        # Initialize target network with the same weights as the policy network
        self.update_target_network()
```

متد update_target_network شبکه هدف را با پارامترهای شبکه Policy به روز می کند.

```python
def update_target_network(self):
    self.target_network.load_state_dict(self.policy_network.state_dict())
```

متد get_q_values مقادیر Q را برای یک وضعیت معین از شبکه سیاست دریافت می کند و اطمینان حاصل می کند که هیچ گرادیانی در طول این عبور رو به جلو محاسبه نمی شود.

```python
def get_q_values(self, state):
    state = torch.tensor(state, dtype=torch.float).unsqueeze(0)
    with torch.no_grad():
        q_values = self.policy_network(state)
    return q_values.numpy()[0]
```

متد choose_action عمل را بر اساس سیاست epsilon-greedy تعیین می کند.

```python
def choose_action(self, state):
    if random.random() < self.exploration_rate:
        return random.randint(0, 7)
    else:
        q_values = self.get_q_values(state)
        return np.argmax(q_values)
```

متد push_to_memory وضعیت، عمل، پاداش، حالت بعدی، انجام شدن بازی را به Replay-Memory انتقال می دهد.

```
def push_to_memory(self, state, action, reward, next_state, done):
    self.replay_memory.push((state, action, reward, next_state, done))
```

متدsample_from_memory از دسته ای از انتقالات از Replay-Memory نمونه می گیرد.

```
def sample_from_memory(self):
    return self.replay_memory.sample(self.batch_size)
```

متد Learn_from_memory تلفات را محاسبه می کند و یک مرحله گرادیان نزولی و بهینه سازی انجام می دهد.

```
def learn_from_memory(self):
    if len(self.replay_memory) < self.batch_size:
        return

    batch = self.sample_from_memory()
    states, actions, rewards, next_states, dones = zip(*batch)

    states = torch.tensor(states, dtype=torch.float)
    actions = torch.tensor(actions, dtype=torch.long)
    rewards = torch.tensor(rewards, dtype=torch.float)
    next_states = torch.tensor(next_states, dtype=torch.float)
    dones = torch.tensor(dones, dtype=torch.float)

    current_q_values = self.policy_network(states).gather(1, actions.unsqueeze(1)).squeeze(1)
    next_q_values = self.target_network(next_states).max(1)[0]
    expected_q_values = rewards + (1 - dones) * self.discount_factor * next_q_values

    loss = F.smooth_l1_loss(current_q_values, expected_q_values)

    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

متدهای train,play, get _cumulative_rewards مانند قسمت طراحی عامل QLearning تعریف می شوند.

```
def train(self, episodes, max_steps_per_episode):
    for episode in range(episodes):
        state = self.env.reset()
        total_reward = 0

        for step in range(max_steps_per_episode):
            action = self.choose_action(state)
            next_state, reward, done = self.env.step(action)
            self.push_to_memory(state, action, reward, next_state, done)

            state = next_state
            total_reward += reward
            self.steps_done += 1

            self.learn_from_memory()

            if done:
                break

        self.cumulative_rewards.append(total_reward)
        self.exploration_rate *= self.exploration_decay

        # Check for consecutive successes
        if reward == 100:  # Reached gold
            self.consecutive_successes += 1
        else:
            self.consecutive_successes = 0

        if self.consecutive_successes == self.consecutive_successes_needed and self.episodes_to_consistency is None:
            self.episodes_to_consistency = episode + 1

        if episode % self.target_update == 0:
            self.update_target_network()

        print(f"Episode {episode + 1}: Total Reward: {total_reward}, Exploration Rate: {self.exploration_rate}")

def play(self, episodes, max_steps_per_episode):
    for episode in range(episodes):
        state = self.env.reset()
        total_reward = 0

        for step in range(max_steps_per_episode):
            action = self.choose_action(state)
            next_state, reward, done = self.env.step(action)
            state = next_state
            total_reward += reward

            if done:
                break

        print(f"Episode {episode + 1}: Total Reward: {total_reward}")

def get_cumulative_rewards(self):
    return self.cumulative_rewards

def get_episodes_to_consistency(self):
    return self.episodes_to_consistency
```

در نهایت محیط WumpusWorldEnv و عامل DQN فراخوانی می شود.عامل را برای تعداد مشخصی اپیزود (۱۰۰۰) و تعداد (۵۰) در هر اپیزود آموزش می دهیم. و در آخر پاداش تجمعی به دست آمده در طول آموزش را ترسیم می کند. سپس بازی را با عامل آموزش دیده برای ۱۰ اپیزود انجام می دهیم.

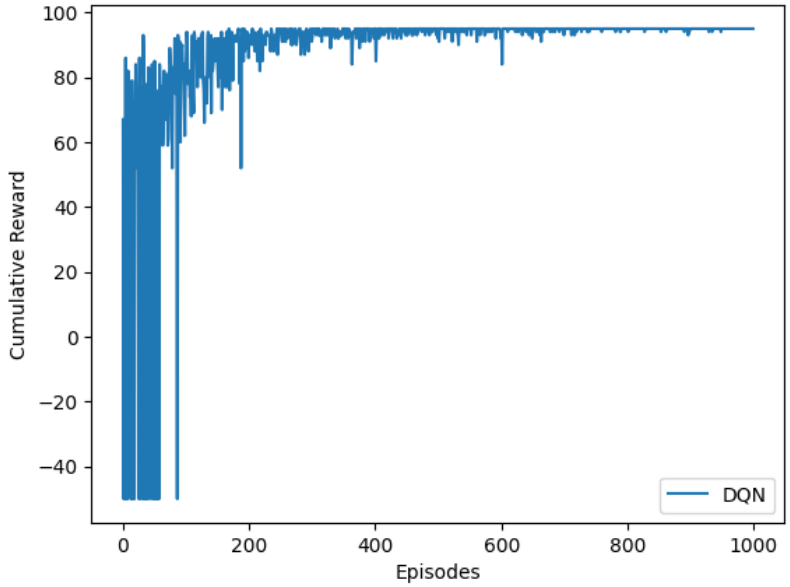
```python
if __name__ == "__main__":
    env = WumpusWorldEnv()

    # Training the DQN agent
    dqn_agent = DQNAgent(env)
    dqn_agent.train(episodes=1000, max_steps_per_episode=50)

    # Plotting the cumulative rewards for the DQN agent
    plt.plot(dqn_agent.cumulative_rewards, label="DQN")
    plt.xlabel('Episodes')
    plt.ylabel('Cumulative Reward')
    plt.legend()
    plt.show()
    # Playing the game with the trained agent
    dqn_agent.play(episodes=10, max_steps_per_episode=50)
```

مقادیر پاداش و نرخ اکتشاف برای اپیزود های اولیه و نهایی به صورت زیر است:

```
Episode 1: Total Reward: -50, Exploration Rate: 0.98
Episode 2: Total Reward: -50, Exploration Rate: 0.9603999999999999
Episode 3: Total Reward: 53, Exploration Rate: 0.9411919999999999
Episode 4: Total Reward: -50, Exploration Rate: 0.9223681599999999
Episode 5: Total Reward: -50, Exploration Rate: 0.9039207967999998
Episode 6: Total Reward: -50, Exploration Rate: 0.8858423808639998
Episode 7: Total Reward: -50, Exploration Rate: 0.8681255332467198
Episode 8: Total Reward: 74, Exploration Rate: 0.8507630225817854
Episode 9: Total Reward: 68, Exploration Rate: 0.8337477621301497
Episode 10: Total Reward: 79, Exploration Rate: 0.8170728068875467
Episode 11: Total Reward: 78, Exploration Rate: 0.8007313507497957
Episode 12: Total Reward: 55, Exploration Rate: 0.7847167237347998
Episode 13: Total Reward: 94, Exploration Rate: 0.7690223892601038
Episode 14: Total Reward: 56, Exploration Rate: 0.7536419414749017
Episode 15: Total Reward: -50, Exploration Rate: 0.7385691026454037
Episode 16: Total Reward: -50, Exploration Rate: 0.7237977205924956
Episode 17: Total Reward: -50, Exploration Rate: 0.7093217661806457
Episode 18: Total Reward: 92, Exploration Rate: 0.6951353308570327
Episode 19: Total Reward: -50, Exploration Rate: 0.6812326242398921
Episode 20: Total Reward: 84, Exploration Rate: 0.6676079717550942
Episode 21: Total Reward: 61, Exploration Rate: 0.6542558123199923
Episode 22: Total Reward: 67, Exploration Rate: 0.6411706960735924
Episode 23: Total Reward: 53, Exploration Rate: 0.6283472821521205
Episode 24: Total Reward: 66, Exploration Rate: 0.6157803365090782
Episode 25: Total Reward: 85, Exploration Rate: 0.6034647297788965
Episode 26: Total Reward: 83, Exploration Rate: 0.5913954351833186
Episode 27: Total Reward: 87, Exploration Rate: 0.5795675264796523
Episode 28: Total Reward: 57, Exploration Rate: 0.5679761759500592
Episode 29: Total Reward: 56, Exploration Rate: 0.5566166524310581
Episode 30: Total Reward: 89, Exploration Rate: 0.5454843193824369
Episode 31: Total Reward: 83, Exploration Rate: 0.5345746329947881
Episode 32: Total Reward: 66, Exploration Rate: 0.5238831403348924
Episode 33: Total Reward: 92, Exploration Rate: 0.5134054775281945
Episode 34: Total Reward: 85, Exploration Rate: 0.5031373679776306
Episode 35: Total Reward: 71, Exploration Rate: 0.493074620618078
Episode 36: Total Reward: 66, Exploration Rate: 0.48321312820571644
Episode 37: Total Reward: 89, Exploration Rate: 0.4735488656416021
Episode 38: Total Reward: 91, Exploration Rate: 0.4640788832877006
Episode 39: Total Reward: 70, Exploration Rate: 0.45479633056219465
Episode 40: Total Reward: 93, Exploration Rate: 0.4457004039509507 3
Episode 41: Total Reward: 93, Exploration Rate: 0.436786395871931 7
Episode 42: Total Reward: 74, Exploration Rate: 0.42805066795449304
Episode 43: Total Reward: 92, Exploration Rate: 0.41948965459540316
Episode 44: Total Reward: 92, Exploration Rate: 0.4110998615034951
```

```
Episode 962: Total Reward: 95, Exploration Rate: 0.008050144525378928
Episode 963: Total Reward: 95, Exploration Rate: 0.008009893802752034
Episode 964: Total Reward: 95, Exploration Rate: 0.007969844333738273
Episode 965: Total Reward: 95, Exploration Rate: 0.007929995112069581
Episode 966: Total Reward: 95, Exploration Rate: 0.007890345136509233
Episode 967: Total Reward: 95, Exploration Rate: 0.007850893410826686
Episode 968: Total Reward: 95, Exploration Rate: 0.007811638943772553
Episode 969: Total Reward: 95, Exploration Rate: 0.00777258074905369
Episode 970: Total Reward: 95, Exploration Rate: 0.007733717845308421 5
Episode 971: Total Reward: 95, Exploration Rate: 0.007695049256081879 5
Episode 972: Total Reward: 95, Exploration Rate: 0.00765657400980147
Episode 973: Total Reward: 95, Exploration Rate: 0.007618291139752462
Episode 974: Total Reward: 95, Exploration Rate: 0.0075801996840537
Episode 975: Total Reward: 95, Exploration Rate: 0.007542298685633431
Episode 976: Total Reward: 95, Exploration Rate: 0.007504587192205264
Episode 977: Total Reward: 95, Exploration Rate: 0.007467064256244237 5
Episode 978: Total Reward: 95, Exploration Rate: 0.007429728934963016
Episode 979: Total Reward: 95, Exploration Rate: 0.007392580290288201
Episode 980: Total Reward: 95, Exploration Rate: 0.00735561738883676
Episode 981: Total Reward: 95, Exploration Rate: 0.007318839301892576
Episode 982: Total Reward: 95, Exploration Rate: 0.007282245105383112 5
Episode 983: Total Reward: 95, Exploration Rate: 0.007245833879856197
Episode 984: Total Reward: 95, Exploration Rate: 0.007209604710456916
Episode 985: Total Reward: 95, Exploration Rate: 0.007173556686904631 5
Episode 986: Total Reward: 94, Exploration Rate: 0.007137688903470108
Episode 987: Total Reward: 94, Exploration Rate: 0.007102000458952757 5
Episode 988: Total Reward: 95, Exploration Rate: 0.007066490456657993 5
Episode 989: Total Reward: 95, Exploration Rate: 0.007031158004374704
Episode 990: Total Reward: 95, Exploration Rate: 0.00699600221435283
Episode 991: Total Reward: 95, Exploration Rate: 0.006961022032810655
Episode 992: Total Reward: 95, Exploration Rate: 0.006926217092264660 5
Episode 993: Total Reward: 95, Exploration Rate: 0.006891586006803337
Episode 994: Total Reward: 95, Exploration Rate: 0.006857128076769321
Episode 995: Total Reward: 95, Exploration Rate: 0.006822842436385474
Episode 996: Total Reward: 95, Exploration Rate: 0.006788728224203546
Episode 997: Total Reward: 95, Exploration Rate: 0.006754784583082528
Episode 998: Total Reward: 95, Exploration Rate: 0.006721010660167116
Episode 999: Total Reward: 95, Exploration Rate: 0.006687405606686628
Episode 1000: Total Reward: 95, Exploration Rate: 0.006653968578831948
```

پاداش تجمعی در طی روند آموزش به صورت زیر رسم شد:



همانطور که از شکل بالا پیداست، در اپیزود های اولیه مقدار پاداش بسیار کم بوده و در طی آموزش و پس از حدود ۲۰۰ اپیزود بهبود یافته و به مقدار 95 همگرا شده است. بنابراین عامل در اثر تعامل با محیط بازی را آموخته است.

نتایج انجام بازی توسط عامل آموزش دیده شده در ۱۰ اپیزود به صورت زیر است:

```
Episode 1: Total Reward: 95
Episode 2: Total Reward: 95
Episode 3: Total Reward: 95
Episode 4: Total Reward: 95
Episode 5: Total Reward: 95
Episode 6: Total Reward: 95
Episode 7: Total Reward: 95
Episode 8: Total Reward: 95
Episode 9: Total Reward: 95
Episode 10: Total Reward: 95
```

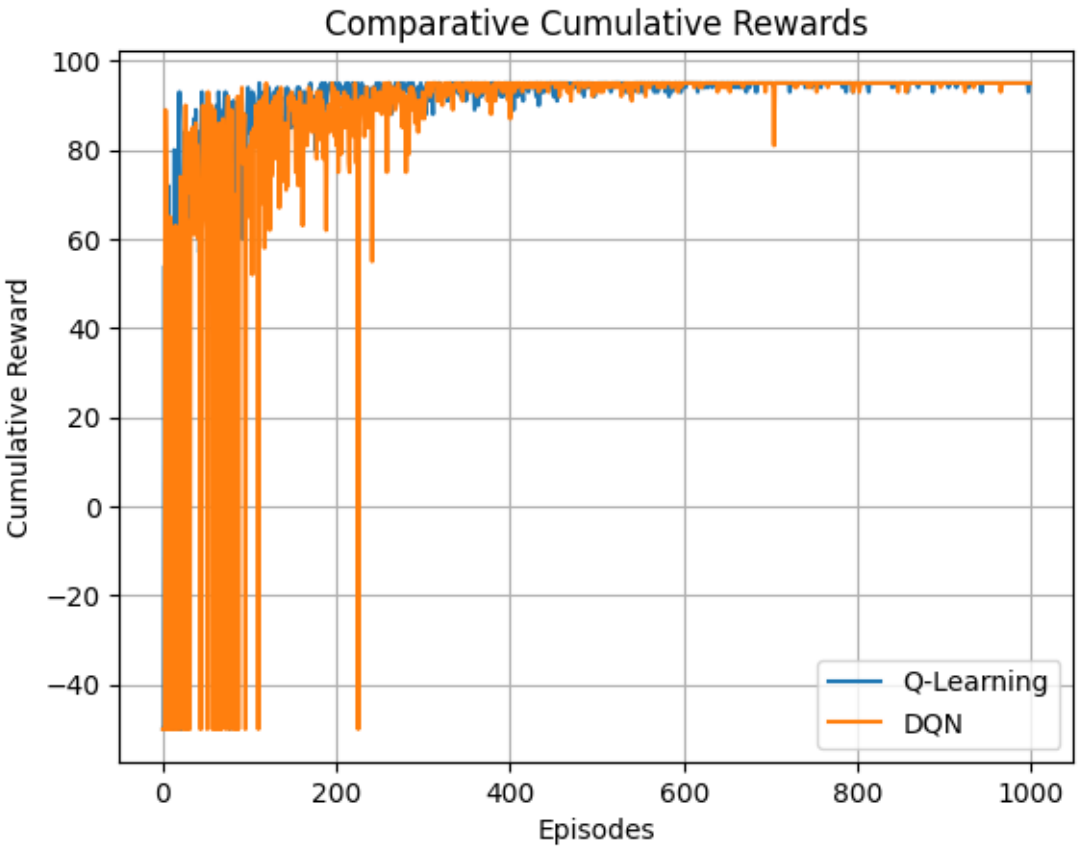
ب عملکرد Policy:

- پاداش تجمعی را در اپیزودها برای هر دو عامل DQN وQ-learning ترسیم کنید .چگونه عملکرد عامل در طول زمان بهبود می یابد؟

پاسخ این قسمت در قسمت الف آمد. در اینجا نیز مجددا عامل ها را آموزش داده و در یک نمودار پاداش تجمعی را رسم می کنیم:

```python
import matplotlib.pyplot as plt

if __name__ == "__main__":
    env = WumpusWorldEnv()

    # Training the Q-Learning agent
    ql_agent = QLearningAgent(env)
    ql_agent.train(episodes=1000, max_steps_per_episode=50)
    ql_cumulative_rewards = ql_agent.get_cumulative_rewards()

    # Training the DQN agent
    dqn_agent = DQNAgent(env)
    dqn_agent.train(episodes=1000, max_steps_per_episode=50)
    dqn_cumulative_rewards = dqn_agent.get_cumulative_rewards()

    # Plotting cumulative rewards for both agents
    plt.plot(ql_cumulative_rewards, label="Q-Learning")
    plt.plot(dqn_cumulative_rewards, label="DQN")
    plt.xlabel('Episodes')
    plt.ylabel('Cumulative Reward')
    plt.title('Comparative Cumulative Rewards')
    plt.legend()
    plt.grid(True)
    plt.show()
```

نتیجه به صورت زیر است:



با افزایش تعداد اپیزودها، هر دو عامل عملکرد بهتری پیدا می کنند و مقدار تجمعی پاداش‌ها افزایش می یابد. در این مسئله DQN دیرتر به سیاست بهینه دست یافته است.

- میانگین پاداش در هر اپیزود را برای هر دو عامل پس از ۱۰۰۰ اپیزود مقایسه کنید .کدام الگوریتم عملکرد بهتری داشت؟

در این قسمت با استفاده از np.mean روی پاداش تجمعی برای هر یک از دو عامل نتیجه را گزارش می کنیم:

برای عامل Q-Learning داریم:

```
import numpy as np

ql_average_reward = np.mean(ql_cumulative_rewards)
print(f"Average reward per episode for Q-Learning: {ql_average_reward}")
```

```
Average reward per episode for Q-Learning: 90.154
```

برای عامل DQN داریم:

```
dqn_average_reward = np.mean(dqn_cumulative_rewards)
print(f"Average reward per episode for DQN: {dqn_average_reward}")
```

```
Average reward per episode for DQN: 86.309
```

از مقایسه میانگین پاداش دو عامل در می یابیم عامل Q-Learning عملکرد بهتری دارد. همچنین می توان از قطعه کد زیر برای بیان نتیجه مقایسه استفاده کرد:

```
if ql_average_reward > dqn_average_reward:
    print("Q-Learning performed better.")
elif ql_average_reward < dqn_average_reward:
    print("DQN performed better.")
else:
    print("Both agents performed equally.")
```

```
Q-Learning performed better.
```

ج

بحث کنید که چگونه نرخ اکتشاف اپسیلون بر فرآیند یادگیری تأثیر می گذارد .وقتی اپسیلون بالا بود در مقابل وقتی کم بود چه چیزی را مشاهده کردید؟

در ابتدای فرآیند نرخ اکتشاف ماکزیمم و برابر یک است و به تدریج با ضریب کاهشی کم می شود. از نتایج قسمت های قبل و مشاهده نمودارهای پاداش تجمعی پیداست، در ابتدای فرآیند عملکرد ضعیف تر

است. اما در طول زمان عملکرد بهبود می یابد. نرخ اکتشاف بالا به عامل اجازه می‌دهد که بیشتر به کاوش محیط بپردازد و به سیاست‌های جدیدتری دست یابد، در حالی که نرخ اکتشاف پایین به عامل اجازه می‌دهد که بیشتر از سیاست‌های موجود استفاده کند که در این حالت احتمال گیر کردن در بهینه محلی زیاد است. بنابراین، با نرخ اکتشاف بالا، ممکن است ابتدا عملکرد ضعیف‌تر باشد اما در طول زمان به سیاست‌های بهتری دست یابد. با نرخ اکتشاف پایین، عامل سریع‌تر به سیاست موجود اعتماد می‌کند و ممکن است به سیاست بهینه نرسد.

د کارایی یادگیری

- چند اپیزود طول کشید تا عاملQ-learning به طور مداوم طلا را بدون افتادن در گودال یا خورده شدن توسط wumpus پیدا کند؟

برای پاسخ به این پرسش باید اولین اپیزودی که پس از آن عملکرد عامل تغییر نمی کند را معرفی کنیم. از قطعه کد زیر استفاده می کنیم:

```
# Printing episodes to consistency
if agent.get_episodes_to_consistency():
    print(f"Agent achieved consistent performance after {agent.get_episodes_to_consistency()} episodes.")
else:
    print("Agent did not achieve consistent performance within the given number of episodes.")
```

```
Agent achieved consistent performance after 58 episodes.
```

بنابراین عامل Q-Learning پس از ۵۸ اپیزود طلا را بدون خورده شدن توسط wumpus یا افتادن در چاله پیدا می کند.

- کارایی یادگیری DQN وQ-learning را مقایسه کنید. کدام یکPolicy بهینه را سریعتر یاد گرفت؟

ابتدا قسمت قبل را برای عامل DQN پیاده سازی و نتایج را گزارش می کنیم:

```
# Printing episodes to consistency
if dqn_agent.get_episodes_to_consistency():
    print(f"Agent achieved consistent performance after {dqn_agent.get_episodes_to_consistency()} episodes.")
else:
    print("Agent did not achieve consistent performance within the given number of episodes.")
```

```
Agent achieved consistent performance after 102 episodes.
```

بنابراین عامل DQN پس از ۱۰۲ اپیزود طلا را بدون خورده شدن توسط wumpus یا افتادن در چاله پیدا می کند.

از نتایج این قسمت و قسمت های قبل مشخص است عامل Q-learning در تعداد اپیزود کمتری سیاست بهینه را پیدا کرده است و نسبت به عامل DQN سریعتر سیاست بهینه را یادگرفته است. در محیط هایی مثل این بازی که پیچیدگی زیادی ندارند، Q-Learning عملکرد بهتری دارد. همچنین وقتی فضای عمل گسسته است از الگوریتم Q-Learning پاسخ بهتری دریافت می شود. و از DQN برای محیط با پیچیدگی بیشتر و اعمال پیوسته استفاده می شود.

۵

معماری شبکه عصبی مورد استفاده برای عامل DQN را شرح دهید .چرا این معماری را انتخاب کردید؟

همانطور که پیش تر توضیح داده شد، شبکه در کلاس QNetwork پیاده سازی شده است.

معماری شبکه عصبی که برای عامل DQN استفاده شده است، با استفاده از سه لایه کاملاً متصل (Fully Connected Layers)طراحی شده است.

لایه ورودی به عنوان ورودی بردار وضعیت (state) را دریافت می‌کند که در محیط بازی اندازه‌ی آن برابر با ابعاد شبکه ۴*۴ است .

لایه میانی با گرفتن خروجی ۶۴ نورون از لایه قبلی، به ۳۲ نورون تبدیل می‌شود که این کاهش اندازه ویژگی‌ها (features) می‌تواند منجر به افزایش سرعت یادگیری شود.

لایه خروجی ۳۲ نورون خروجی لایه قبلی را گرفته و به تعداد عمل های ممکن (۸ عمل) تبدیل می‌کند.

با توجه به اینکه ورودی این لایه ویژگی‌های پردازش شده از وضعیت فعلی است، خروجی این لایه می‌تواند Q-value برای هر عمل را تخمین بزند، که عامل بر اساس آن تصمیم‌گیری می‌کند.

دلایل انتخاب این معماری:

راحتی در پیاده سازی شبکه های متصل، عملکرد مطلوب شبکه عصبی در صورت انتخاب مناسب نورون، تخمین مطلوب در انواع مسائل. در اینجا از شبکه عصبی برای تخمین Q-Value و معرفی بهترین عمل در هر حالت استفاده شد.

مراجع

[1] https://github.com/MJAHMADEE/MachineLearning2024W