دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

پروژه مبانی هوش مصنوعی

گزارش کتبی فاز دوم

اعضای گروه:

فرناز موحدی

ارشیا شفیعی

رضا چراخ

آذر ۱۴۰۳

فهرست مطالب

عنوان **صفحه**

[1] "Youtube," [Online]. Available: https://youtu.be/gqC_p2XWpLU?si=X0FYGh9eSTlYnUAy.

[2] "Youtube," [Online]. Available: https://youtu.be/phgI_880uSw?si=A0lQlqt_R_wb29Ro.

[3] "Youtube," [Online]. Available: https://youtu.be/9g32v7bK3Co?si=oE5DLe-H4e_X3P-a.

[4] "Youtube," [Online]. Available: https://youtu.be/l87rgLg90HI?si=Fb7NkDsUW755s1Ow.

[5] "Youtube," [Online]. Available: https://youtu.be/Jk2V9yA82YU?si=Y1Nn5lTXMsiz45rK.

فهرست مطالب

عنوان **صفحه**

[6] "Youtube," [Online]. Available: https://youtu.be/3Rx2x2traxw?si=TAUXXCCKh2SHWH-a.

[7] "Youtube," [Online]. Available: https://youtu.be/4KGC_3GWuPY?si=QJeL0v9FLHMlyBna.

[8] "ChatGPT OpenAI," [Online]. Available: https://chat.openai.com.

فصل اول
فرآیند تصمیم مارکوف

۱-۱- مقدمه

فصل اول گزارش، به حل مسئله‌ای مبتنی بر فرآیند تصمیم‌گیری مارکوف می‌پردازد که در آن، یک عامل (پرنده خشمگین) در محیطی تصادفی و مشاهده پذیر تلاش می‌کند از نقطه شروع به هدف برسد. عامل با انتخاب کنش‌های بالا، پایین، راست و چپ، که احتمال موفقیت آن‌ها تصادفی است، بهینه‌ترین مسیر را برای حداکثرسازی پاداش خود می‌یابد. هدف پروژه، یافتن سیاست بهینه و بررسی عملکرد عامل در چندین اجرای مختلف محیط است.

۱-۲- تابع reward function

۱-۲-۱- شرح تابع

تابع reward_function یک نقشه پاداش (reward map) ایجاد می‌کند که وضعیت‌های محیط را بر اساس نوع سلول و موقعیت عامل بررسی می‌کند. این نقشه بهینه‌سازی تصمیمات عامل را در فازهای مختلف، مانند تمرکز بر خوک‌ها یا رسیدن به هدف، ممکن می‌سازد و در هر فاز، تنها یک هدف دارد. این تابع اولویت را به نزدیک‌ترین هدف (می‌تواند خوک یا تخم مرغ‌ها باشد) می‌دهد و بقیه سلول‌ها از ارزش کمتری برخوردار هستند.

از آنجایی که هدف نهایی رسیدن به حداقل امتیاز است، در ابتدا این تابع با پارامتر phase = pigs فراخوانی می‌شود. در این حالت نزدیک‌ترین خوک تبدیل به هدف می‌شود و پاداش زیادی به آن اختصاص می‌یابد. هربار که تعداد خوک‌های خورده‌شده یکی افزایش می‌یابد، این تابع دوباره صدا زده می‌شود تا دوباره نزدیک‌ترین خوک به عامل در آن لحظه محاسبه شود. پس از ازبین بردن تعداد مشخصی خوک، این تابع با پارامتر phase = goal فراخوانی می‌شود که تمرکز آن رسیدن به هدف یا همان تخم مرغ‌ها است.

یک نقشه‌ی پاداش اولیه با مقادیر پیش‌فرض (DEFAULT_REWARD) ساخته می‌شود که اندازه آن

برابر با شبکه‌ی محیط ۸×۸ است.

موقعیت فعلی عامل (پرنده) از ویژگی self.__agent_pos دریافت می‌شود.

موقعیت تمام سلول‌هایی که شامل خوک (P) هستند در یک لیست ذخیره می‌شود.

تابع نزدیک‌ترین خوک به عامل را پیدا می‌کند. فاصله به‌صورت فاصله منهتن محاسبه می‌شود:

$$\text{Manhattan Distance} = |X1 - X2| + |Y1 - Y2|$$

تابع بر اساس نوع هر سلول مانند (G, P, Q, T, R) پاداش‌ها را تنظیم می‌کند.

اگر سلول هدف باشد:

- در حالت goal پاداش بسیار بالا (۲۰۰۰۰) برای رسیدن به هدف داده می‌شود.
- در سایر حالات، پاداش کمی (۱۰) اختصاص می‌یابد.

اگر سلول حاوی خوک باشد:

- در حالت pigs
  - نزدیک‌ترین خوک پاداش بالایی (۲۰۰۰۰) دریافت می‌کند.
  - سایر خوک‌ها پاداش کوچکی (۱۰) دریافت می‌کنند.
- در سایر حالات، تمام خوک‌ها پاداش ۱۰ دریافت می‌کنند.

اگر سلول حاوی ملکه خوک باشد، جریمه‌ی سنگینی (۲۰۰۰-) به عامل وارد می‌شود.

سلول‌های معمولی و سنگ‌ها مقدار جریمه‌ی کوچکی (۱۰-) دریافت می‌کنند تا عامل از ورود به آن‌ها اجتناب کند و تا جای ممکن از کنش‌های بی فایده جلوگیری شود

در نهایت، تابع نقشه‌ی پاداش جدید را بازمی‌گرداند. این نقشه در ادامه توسط تابع value iteration استفاده می‌شود.

۱-۲-۲- کد تابع

```
def reward_function(self, phase='pigs'):  3 usages (2 dynamic)  farnaz +1 *
    """
    reward_map = [[DEFAULT_REWARD for _ in range(self.__grid_size)] for _ in range(self.__grid_size)]

    # Get agent position
    agent_pos = self.__agent_pos

    # Track all pig positions
    pigs_positions = [(row, col) for row in range(self.__grid_size) for col in range(self.__grid_size) if
                      self.grid[row][col] == 'P']

    # Find the nearest pig to the agent
    if pigs_positions:
        nearest_pig = min(pigs_positions, key=lambda pig: abs(pig[0] - agent_pos[0]) + abs(pig[1] - agent_pos[1]))
    else:
        nearest_pig = None

    # Assign rewards to grid cells
    for row in range(self.__grid_size):
        for col in range(self.__grid_size):
            cell = self.grid[row][col]

            if cell == 'G':  # Goal
                reward_map[row][col] = 20000 if phase == 'goal' else 10
            elif cell == 'P':
                if phase == 'pigs':
                    if (row, col) == nearest_pig:
                        reward_map[row][col] = 20000  # High reward for the nearest pig
                    else:
                        reward_map[row][col] = 10  # Smaller reward for other pigs
                else:
                    reward_map[row][col] = 10
            elif cell == 'Q':  # Queen pig
                reward_map[row][col] = -2000
            elif cell == 'T':
                reward_map[row][col] = -10
            elif cell == 'R':  # Rock
                reward_map[row][col] = -10

    return reward_map
```

۱-۳- تابع value iteration

تابع value_iteration یک الگوریتم برای حل مسئله فرآیند تصمیم‌گیری مارکوفی (MDP) است. این الگوریتم با استفاده از تکرار مقدار (Value Iteration)، بهترین سیاست (policy) و یک Heat Map که نشان دهنده v_value به ازای هر خانه از محیط بازی می‌باشد را برمی‌گرداند.

۱-۳-۱- ورودی‌ها

env: محیط بازی که شامل شبکه (grid) و توابع مرتبط مانند reward_function است.

transition_table: جدول انتقال که احتمال، وضعیت بعدی و پاداش برای هر حرکت را ذخیره

می‌کند.

discount_factor: نرخ کاهش (معمولاً عددی بین ۰ و ۱) که میزان اهمیت پاداش‌های آینده را تعیین می‌کند. پیش‌فرض: 0.9

theta: مقدار آستانه برای همگرایی الگوریتم است. وقتی تغییرات مقادیر حالت‌ها از این مقدار کمتر شود، الگوریتم متوقف می‌شود. پیش‌فرض: 10^-7

phase: فاز فعلی مسئله (مانند pigs یا goal) که تعیین می‌کند تابع پاداش چگونه تولید شود.

۱-۳-۲- متغیرهای اولیه

V: دیکشنری value function که مقدار هر حالت را نگهداری می‌کند. در ابتدا همه مقادیر صفر هستند.

policy: دیکشنری سیاست (policy) که کنش بهینه برای هر حالت را نگهداری می‌کند.

rewards: نقشه پاداش تولیدشده توسط تابع reward_function برای حالت جاری (phase) است.

delta_history: لیستی که تغییرات مقادیر (Δ\DeltaΔ) را در طول تکرارها ذخیره می‌کند.

۱-۳-۳- حلقه اصلی الگوریتم

این حلقه تا زمانی که مقادیر همگرا شوند (Δ<theta) ادامه پیدا می‌کند.

Δ نشان‌دهنده بیشترین تغییر در مقدار هر حالت در طول تکرار فعلی است. این مقدار برای بررسی همگرایی استفاده می‌شود.

دو حلقه تمام سلول‌های محیط (۸×۸) را پیمایش می‌کند. اگر حالت فعلی شامل سنگ (R) باشد، آن را نادیده می‌گیرد.

حلقه اول چهار کنش ممکن (بالا، پایین، چپ، راست) را بررسی می‌کند و برای هریک از آنها مقدار موردانتظاری را محاسبه می‌کند. در ادامه طبق فرمول بلمن در value iteration، مقدار بیشینه آنها انتخاب می‌شود و این مقدار به‌عنوان V(state) به‌روزرسانی می‌شود. فرمول بلمن:

$$V(s) = \max_a(R(s,a) + \gamma V(s'))$$

تغییر delta محاسبه شده و مقدار بیشینه تغییر ذخیره می‌شود. سیاست (π) برای هر حالت با انتخاب عمل بهینه (argmax) به‌روزرسانی می‌شود. اگر مقدار بیشینه تغییر (Δ) کمتر از آستانه (theta) باشد، الگوریتم متوقف می‌شود. پس از تکمیل تکرار مقدار، سیاست بهینه توسط تابع correct_policy اصلاح

می‌شود تا کنش‌های نامعتبر حذف شوند و بهترین کنش معتبر انتخاب شود.

۱-۳-۴- خروجی تابع

corrected_policy: سیاست اصلاح‌شده که حالت‌ها را به کنش‌های بهینه نگاشت می‌کند.

V: مقدار نهایی هر حالت.

delta_history: تاریخچه تغییرات برای تحلیل همگرایی.

۱-۳-۵- کد تابع

```python
def value_iteration(env, transition_table, discount_factor=0.9, theta=1e-7, phase='pigs'):  4 usages  farnaz +1 *
    """
    V = defaultdict(float)  # Value function initialized to 0
    policy = {}  # Policy to store optimal actions
    rewards = env.reward_function(phase)  # 2D list of rewards
    delta_history = []

    while True:
        delta = 0

        for x in range(8):
            for y in range(8):
                state = (x, y)
                if env.grid[x][y] == "R":  # Skip states with rocks
                    continue

                action_values = []
                for action in range(4):  # Up, Down, Left, Right
                    value = 0
                    for prob, next_state, _ in transition_table[state][action]:
                        next_x, next_y = next_state
                        reward = rewards[next_x][next_y]  # Access reward using 2D indices
                        value += prob * (reward + discount_factor * V[next_state])
                    action_values.append(value)

                # Update value and policy
                best_action_value = max(action_values) if action_values else 0
                delta = max(delta, abs(V[state] - best_action_value))
                V[state] = best_action_value
                if action_values:
                    policy[state] = np.argmax(action_values)

        delta_history.append(delta)

        # Stop if the values converge
        if delta < theta:
            break

    corrected_policy = correct_policy(policy, V, env, discount_factor)

    return corrected_policy, V, delta_history
```

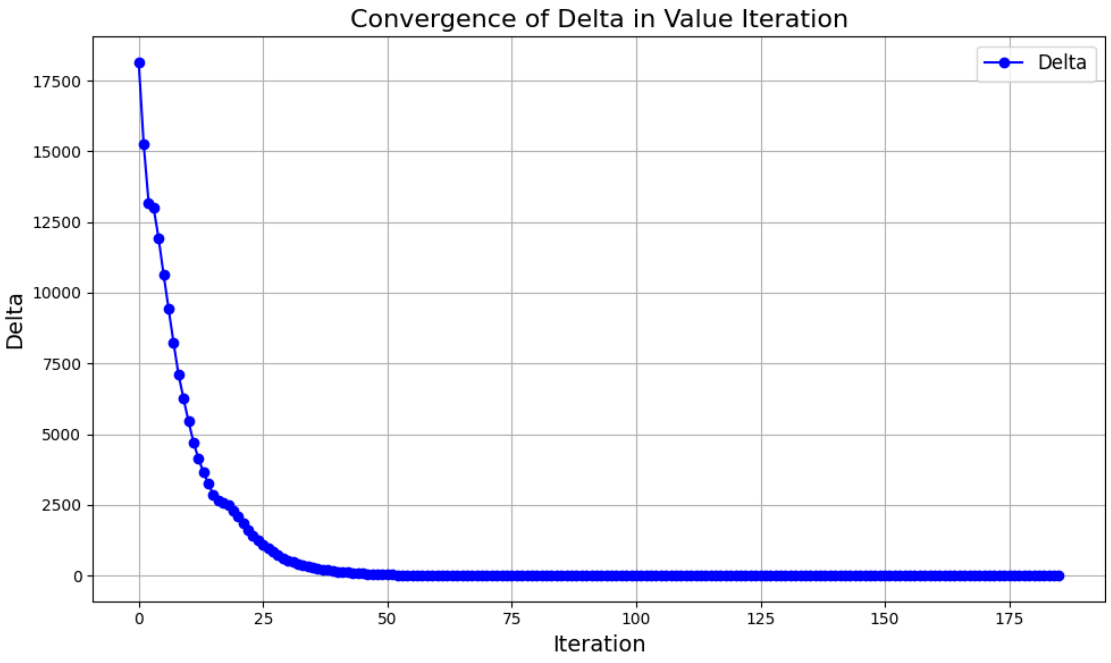
۱-۴- تابع correct policy

تابع correct_policy برای آن است که کنش‌های نامعتبر در حاشیه‌های محیط بررسی شوند. این تابع سیاست ورودی را با بررسی محدودیت‌های محیط و تابع ارزش بهینه می‌کند. ابتدا سیاست کپی می‌شود تا تغییرات روی نسخه‌ای مستقل اعمال گردد. سپس برای هر حالت، موقعیت آن در grid بررسی شده و بر اساس اینکه در لبه‌ها است یا وسط grid، مجموعه‌ای از کنش‌های مجاز تعیین می‌شود. برای هر کنش مجاز، ارزش انتظاری آن با استفاده از احتمالات انتقال، پاداش حالت بعدی، و مقدار گاما محاسبه می‌گردد. کنشی که بالاترین ارزش انتظاری را دارد به‌عنوان بهترین کنش انتخاب شده و سیاست برای آن حالت به‌روزرسانی می‌شود. در نهایت، سیاست اصلاح‌شده که کنش‌های غیرمعتبر را حذف کرده است، بازگردانده می‌شود.

۱-۵- ارزیابی

۱-۵-۱- تحلیل همگرایی delta

همانطور که انتظار می‌رود، مقادیر delta با پیش رفتن تکرارها کاهش یافته و به صفر میل می‌کنند.



۱-۵-۲- تحلیل heat map و سیاست بهینه

این موارد برای حالتی که value iteration با phase = goal فراخوانی می‌شود، نشان داده شده است. درحالتی که محیط به صورت زیر باشد، سیاست بهینه به این صورت خواهد بود.

```
Optimal Policy:
↓ R → ← R ↓ → ↓
↓ ↓ ↓ R → ↓ → ↓
↓ → ↓ ↓ ↓ → ↓ ↓
↑ ↑ → → → → ↓ ↓
↑ → → → → → ↓ ↑
↓ ↑ ↑ → ↑ → ↓ R
↑ ↑ ← ↑ R R ↓ R
↓ → ↑ ← ← R → ↓
```

```
T R T P R T T T
T T P R T T T T
T T P P T Q T T
T T T T T T T T
T T T T T T T T
Q T T P T T T R
P P T T R R T R
T T T P T R T G
```

همچنین heat map برای این اجرای تابع value iteation به صورت زیر خواهد بود. همانطور که انتظار می‌رود، این مقادیر در هدف و نزدیکی هدف بسیار زیاد و در بقیه نواحی کمتر است.

State Values (V_goal):

	0	1	2	3	4	5	6	7
0	16806.93	0.00	22291.13	14321.10	0.00	36894.49	43053.58	43844.08
1	22857.57	25305.09	28792.83	0.00	34783.11	48017.20	49023.32	50082.28
2	28864.17	31840.31	37582.79	41147.95	44363.65	52201.80	58652.33	59593.88
3	29367.30	30922.68	38649.93	45966.14	52229.07	60057.46	63412.97	58185.91
4	29201.69	36614.88	45378.19	47652.97	53000.54	62090.58	77863.63	64922.42
5	25549.78	29703.58	34603.75	44789.30	52329.84	63737.48	95394.42	0.00
6	24766.09	28271.89	28425.56	30016.97	0.00	0.00	135230.86	0.00
7	23708.02	25863.08	28882.44	24725.68	20195.55	0.00	224481.73	239184.78

فصل دوم
محیط‌های ناشناخته

۲-۱- مقدمه

این کد مربوط به پیاده‌سازی الگوریتم Q-Learning برای یادگیری یک عامل (Agent) در محیطی خاص است که به عنوان "QLearningAgent" معرفی شده است. هدف این الگوریتم، یادگیری بهترین سیاست برای انجام اعمال مختلف در یک محیط است به طوری که بیشترین پاداش ممکن را برای عامل به ارمغان آورد. در این کد، عامل از جدول Q (Q-table) برای یادگیری استفاده می‌کند و برای دو نوع پاداش متفاوت، دو جدول Q مجزا دارد: یکی برای تعاملات مربوط به خوک‌ها و دیگری برای رسیدن به هدف نهایی.

۲-۲- Q learning

انتخاب اقدام: عامل ابتدا یک حالت از محیط دریافت می‌کند و بر اساس سیاست خود (حساس به خوک‌ها یا هدف) از جدول Q مناسب اقدام انتخاب می‌کند.

اجرای اقدام: اقدام انتخاب‌شده در محیط اجرا می‌شود و پاداش (reward) دریافت می‌شود.

به‌روزرسانی جدول Q: پس از دریافت پاداش، جدول Q برای حالت و اقدام انتخاب‌شده به‌روزرسانی می‌شود تا سیاست بهتری برای آینده پیدا کند.

کاهش نرخ اکتشاف: با گذشت زمان، عامل از اکتشاف تصادفی کمتر استفاده می‌کند و به سیاست‌های خود تکیه بیشتری می‌کند.

```
def __init__(self, state_size, action_size, learning_rate=0.1, discount_factor=0.99, exploration_decay=0.99, min_exploration_rate=0.3):
    self.state_size = state_size
    self.action_size = action_size
    self.q_table_pig = np.zeros((state_size, action_size))
    self.q_table_goal = np.zeros((state_size, action_size))
    self.learning_rate = learning_rate
    self.discount_factor = discount_factor
    self.exploration_rate = 1.0
    self.exploration_decay = exploration_decay
    self.min_exploration_rate = min_exploration_rate
```

۲-۲-۱- شرح تابع update q table

در این تابع استیت‌های q table را بر اساس بهترین کنش و reward متناظر با آن استیت بروزرسانی می‌کنیم.

```
def update_goal_base_q_table(self, state, action, reward, next_state, done):
    best_next_action = np.argmax(self.q_table_goal[next_state])
    target = reward + self.discount_factor * self.q_table_goal[next_state][best_next_action] * (1 - done)
    self.q_table_goal[state][action] += self.learning_rate * (target - self.q_table_goal[state][action])

def update_pig_base_q_table(self, state, action, reward, next_state, done):
    best_next_action = np.argmax(self.q_table_pig[next_state])
    target = reward + self.discount_factor * self.q_table_pig[next_state][best_next_action] * (1 - done)
    self.q_table_pig[state][action] += self.learning_rate * (target - self.q_table_pig[state][action])
```

۲-۲-۲- شرح تابع Choose action

این متد برای انتخاب اقدام مناسب از جدول Q استفاده می‌شود:

اگر یک عدد تصادفی کوچک‌تر از نرخ اکتشاف باشد، عامل اقدام تصادفی (explore) انجام می‌دهد.

اگر سیاست "pig" باشد (یعنی هدف نزدیک به خوک‌ها باشد)، از q_table_pig برای انتخاب اقدام استفاده می‌کند.

اگر سیاست "goal" باشد (یعنی هدف رسیدن به هدف باشد)، از q_table_goal برای انتخاب اقدام استفاده می‌کند..

```python
def choose_action(self, state, policy):
    if np.random.rand() < self.exploration_rate:
        return np.random.choice(self.action_size)  # Explore
    if (policy == "pig"):
        return np.argmax(self.q_table_pig[state])  # Exploit
    elif (policy == "goal"):
        return np.argmax(self.q_table_goal[state])  # Exploit
```

decay_exploration –۳–۲–۲

در این متد، نرخ اکتشاف عامل کاهش می‌یابد تا عامل بیشتر به سیاست‌های خود اعتماد کند و از اکتشاف تصادفی کمتر استفاده کند. این نرخ تا حداقل مقدار مشخص شده (min_exploration_rate) کاهش می‌یابد.

```
def decay_exploration(self):
    self.exploration_rate = max(self.min_exploration_rate, self.exploration_rate * self.exploration_decay)
```

reward mapping -۴-۲-۲

این دو متد مسئول تبدیل پاداش‌ها به مقادیر نهایی برای هر سیاست هستند که طی آزمون و خطاها و تجربه هایی بدست آمده است.

```
def reward_mapping_goal_base(self, reward):
    if (reward == QLearningAgent.DEFAULT_REWARD):
        return -10
    elif (reward == QLearningAgent.GOOD_PIG_REWARD):
        return 400
    elif (reward == QLearningAgent.QUEEN_REWARD):
        return -800
    elif (reward == QLearningAgent.TNT_REWARD):
        return -2000
    elif (reward == QLearningAgent.GOAL_REWARD):
        return 4000
    elif (reward == QLearningAgent.ACTION_TAKEN_REWARD):
        return -1000

def reward_mapping_pig_base(self, reward):
    if (reward == QLearningAgent.DEFAULT_REWARD):
        return -10
    elif (reward == QLearningAgent.GOOD_PIG_REWARD):
        return 1000
    elif (reward == QLearningAgent.QUEEN_REWARD):
        return -900
    elif (reward == QLearningAgent.TNT_REWARD):
        return -3500
    elif (reward == QLearningAgent.GOAL_REWARD):
        return 1100
    elif (reward == QLearningAgent.ACTION_TAKEN_REWARD):
        return -1000
```

main -۵-۲-۲

در این محیط، عامل با دو نوع هدف اصلی روبه‌رو است:

یادگیری نحوه تعامل با خوک‌ها (که هر خوک نمایانگر یک هدف است).

یادگیری نحوه رسیدن به هدف اصلی (که در اینجا رسیدن به نقطه‌ای خاص است).

در این پیاده‌سازی، از دوقسمت جدول Q برای یادگیری در دو زمینه جداگانه (خوک‌ها و هدف) استفاده می‌شود.

کد به سه بخش اصلی تقسیم می‌شود:

آموزش مربوط به خوک‌ها: در این مرحله، عامل یاد می‌گیرد چگونه خوک‌ها را پیدا کند و از بین ببرد.

آموزش مربوط به هدف: در این مرحله، عامل به سمت هدف حرکت می‌کند.

ارزیابی: پس از اتمام آموزش، عامل مورد ارزیابی قرار می‌گیرد و نتایج آن نمایش داده می‌شود.

```
class SGD:
    Farnaz Movahedi *
    def __init__(self, lr=0.001, epochs=1000, batch_size=1024, tol=1e-3, momentum=0.9):
        self.learning_rate = lr
        self.epochs = epochs
        self.batch_size = batch_size
        self.tolerance = tol
        self.momentum = momentum
        self.weights = None
        self.bias = None
        self.velocity_weights = None
        self.velocity_bias = None
```

۲-۲-۵-۱- آموزش مربوط به بخش خوک‌ها

در این بخش، عامل به یادگیری نحوه تعامل با خوک‌ها می‌پردازد:

در هر اپیزود از آموزش، محیط ریست می‌شود و عامل به حرکت خود ادامه می‌دهد تا زمانی که هدف یا پایان بازی تحقق یابد.

عامل با استفاده از جدول Q مخصوص خوک‌ها (q_table_pig) اقدام انتخابی انجام می‌دهد.

اگر عامل به خوک برسد، پاداشی که از طریق متد reward_mapping_pig_base محاسبه می‌شود، اضافه می‌شود و جدول Q به‌روزرسانی می‌شود.

همچنین، نرخ اکتشاف (exploration rate) در هر مرحله کاهش می‌یابد.

```python
pig_eaten_states = set()  # Track states where pigs have been eaten

# Parameters for convergence
epsilon = 0.001
value_differences = []

# Training phase for pigs
num_training_episodes = 1000
for episode in range(num_training_episodes):
    state = env.reset()
    done = False

    # Backup Q-table for difference calculation
    old_q_table = agent.q_table_pig.copy()

    while not done:
        state_index = state[0] * env._UnknownAngryBirds__grid_size + state[1]
        action = agent.choose_action(state_index, "pig")
        next_state, reward, pig_state, done = env.step(action)
        reward = agent.reward_mapping_pig_base(reward)

        last_pig_states = pig_state
        next_state_index = next_state[0] * env._UnknownAngryBirds__grid_size + next_state[1]

        # Update eaten pig states
        if reward == QLearningAgent.GOOD_PIG_REWARD:
            pig_eaten_states.add(next_state)

        agent.update_pig_base_q_table(state_index, action, reward, next_state_index, done)

        state = next_state
    agent.decay_exploration()

    # Calculate Value Difference
    value_difference = np.sum(np.abs(agent.q_table_pig - old_q_table))
    value_differences.append(value_difference)
    print(f"Episode {episode + 1}, Value Difference: {value_difference}")

    # Check for convergence
    if value_difference < epsilon:
        print(f"Converged after {episode + 1} episodes.")
        break

print("Training pig based completed. Starting goal-based training...")
```

۲-۲-۵-۲- بخش آموزش برای رسیدن به هدف

در این بخش، عامل یاد می‌گیرد که چگونه به هدف نهایی برسد:

مشابه بخش قبلی، در این بخش نیز محیط ریست می‌شود و عامل به سمت هدف حرکت می‌کند.

عامل از جدول Q مخصوص هدف (q_table_goal) برای انتخاب اقدامات استفاده می‌کند.

در اینجا نیز همانند آموزش خوک‌ها، جدول Q برای اهداف به‌روزرسانی می‌شود و در هر مرحله نرخ

اکتشاف کاهش می‌یابد.

۲-۲-۵- ارزیابی

پس از اتمام آموزش‌ها، عامل در چندین اپیزود ارزیابی می‌شود:

در این مرحله، نرخ اکتشاف غیرفعال می‌شود (agent.exploration_rate = 0) و عامل تنها از سیاست‌هایی که آموخته است استفاده می‌کند.

در هر اپیزود، عامل به محیط وارد می‌شود و تلاش می‌کند تا بهترین عملکرد را با استفاده از جدول Qآموخته شده برای خوک‌ها و هدف انجام دهد.

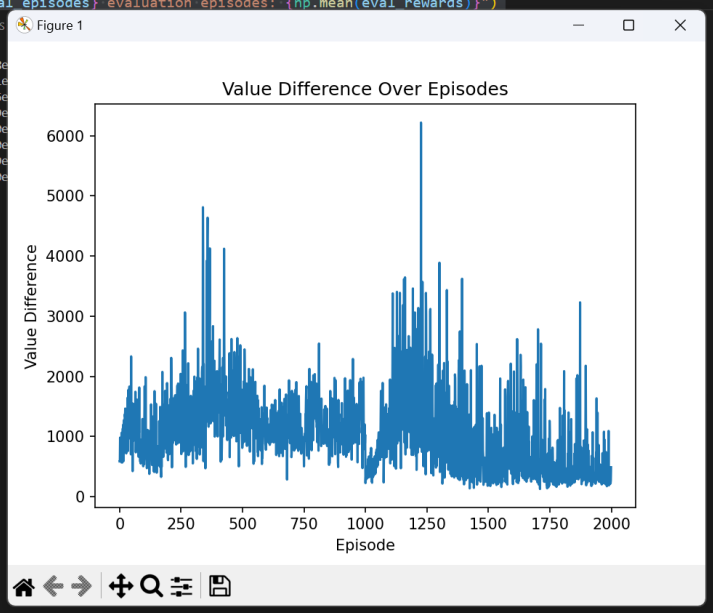
پاداش نهایی محاسبه شده و نمایش داده می‌شود.

```
# Evaluation phase
agent.exploration_rate = 0  # Turn off exploration for evaluation
num_eval_episodes = 5
eval_rewards = []

print(agent.q_table_pig)
for episode in range(num_eval_episodes):
    state = env.reset()
    total_reward = 0
    done = False

    moves = 0
    while not done:
        moves += 1
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()

        env.render(screen)

        state_index = state[0] * env._UnknownAngryBirds__grid_size + state[1]
        num_eaten_pigs = sum(1 for s in pig_state if s)

        if total_reward < 700 and moves < 100:
            if 20 < moves < 40 and num_eaten_pigs < 1:
                action = agent.choose_action(state_index, "goal")  # Switch to goal if no pigs eaten
            else:
                action = agent.choose_action(state_index, "pig")  # Focus on pigs
        else:
            action = agent.choose_action(state_index, "goal")  # Focus on goal

        next_state, reward, pig_state, done = env.step(action)

        state = next_state
        total_reward += reward

        pygame.display.flip()
        clock.tick(FPS)

    eval_rewards.append(total_reward)
    print(f"Evaluation Episode {episode + 1}: Reward = {total_reward}")

print(f"Average reward over {num_eval_episodes} evaluation episodes: {np.mean(eval_rewards)}")
```

value difference and policy grid –۵–۲–۲

```python
# Visualize Policy
policy_grid = np.empty((env._UnknownAngryBirds__grid_size, env._UnknownAngryBirds__grid_size), dtype=str)
for x in range(env._UnknownAngryBirds__grid_size):
    for y in range(env._UnknownAngryBirds__grid_size):
        state_index = x * env._UnknownAngryBirds__grid_size + y
        best_action = np.argmax(agent.q_table_pig[state_index])
        if best_action == 0:
            policy_grid[x, y] = 'U'
        elif best_action == 1:
            policy_grid[x, y] = 'D'
        elif best_action == 2:
            policy_grid[x, y] = 'L'
        elif best_action == 3:
            policy_grid[x, y] = 'R'

print("Policy Grid:")
print(policy_grid)

pygame.quit()

# Plot Value Difference
import matplotlib.pyplot as plt

plt.plot(value_differences)
plt.title("Value Difference Over Episodes")
plt.xlabel("Episode")
plt.ylabel("Value Difference")
plt.show()
```

```
135
136     print(f"Average reward over {num_eval_episodes} evaluation episodes: {np.mean(eval_rewards)}")
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

 [ 1.59834506e+03  4.43099877e+02  4.57481622e+02  2.27994568e
 [ 1.63942597e+03  5.13806188e+02  2.76210135e+01  7.89114811e
 [ 1.74638348e+03  7.49281184e+02  9.51380144e+02  9.53081336e
 [ 7.26416709e+02  8.25835201e+02  6.09496852e+02  7.11667299e
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e
 [ 5.28992813e+02  5.64798445e+02  3.38604012e+02  1.20001869e
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e
Evaluation Episode 1: Reward = 1791
Evaluation Episode 2: Reward = 1792
Evaluation Episode 3: Reward = 1789
Evaluation Episode 4: Reward = 1786
Evaluation Episode 5: Reward = 1796
Average reward over 5 evaluation episodes: 1790.8
Policy Grid:
[['L' 'L' 'U' 'D' 'L' 'R' 'R' 'D']
 ['D' 'U' 'U' 'D' 'L' 'U' 'D' 'L']
 ['U' 'U' 'U' 'L' 'L' 'U' 'D' 'L']
 ['R' 'L' 'U' 'R' 'U' 'U' 'D' 'D']
 ['U' 'R' 'R' 'R' 'L' 'L' 'D' 'L']
 ['U' 'U' 'R' 'U' 'D' 'U' 'U' 'U']
 ['U' 'L' 'U' 'U' 'U' 'L' 'D' 'U']
 ['U' 'U' 'U' 'D' 'U' 'U' 'R' 'U']]
```



منابع

[1] "Youtube [متصل]. "، Available: https://youtu.be/gqC_p2XWpLU?si=X0FYGh9eSTlYnUAy.

[2] "Youtube [متصل]. "، Available: https://youtu.be/phgI_880uSw?si=A0lQlqt_R_wb29Ro.

[3] "Youtube [متصل]. "، Available: https://youtu.be/9g32v7bK3Co?si=oE5DLe-H4e_X3P-a.

[4] "Youtube [متصل]. "، Available: https://youtu.be/l87rgLg90HI?si=Fb7NkDsUW755s1Ow.

[5] "Youtube [متصل]. "، Available: https://youtu.be/Jk2V9yA82YU?si=Y1Nn5lTXMsiz45rK.

[6] "Youtube [متصل]. "، Available: https://youtu.be/3Rx2x2traxw?si=TAUXXCCKh2SHWH-a.

[7] "Youtube [متصل]. "، Available: https://youtu.be/4KGC_3GWuPY?si=QJeL0v9FLHMlyBna.

[8] "ChatGPT OpenAI [متصل]. "، Available: https://chat.openai.com.