به نام خدا

گزارش کار فاز دوم پروژه

فرایند تصمیم مارکوف (MDP)

درس: مبانی و کاربرد هوش مصنوعی

استاد درس: دکتر کارشناس

دستیار آموزشی: پویا صامتی

اعضای تیم :

یونس ایوبی راد - پویا اسفندانی

پیاده سازی تابع Reward function

```python
def reward_function(self):
    current_state = self.current_state
    grid = self.grid
    reward_map = [[0 for _ in range(self.__grid_size)] for _ in range(self.__grid_size)]

    x_start, y_start = current_state
    pig_positions = []

    for i in range(self.__grid_size):
        for j in range(self.__grid_size):
            if grid[i][j] == 'P':
                pig_positions.append((i, j))
            elif grid[i][j] == 'Q':
                if (i, j) != (6, 6) or (7, 6) or (6, 7):
                    reward_map[i][j] = -0.5
                else:
                    reward_map[i][j] = -0.05
            elif grid[i][j] == 'R':
                reward_map[i][j] = -0.01

    if pig_positions:
        min_distance = float('inf')
        nearest_pig = None
        for px, py in pig_positions:
            distance = abs(x_start - px) + abs(y_start - py)
            if distance < min_distance:
                min_distance = distance
                nearest_pig = (px, py)

        px, py = nearest_pig
        reward_map[px][py] = 1
    else:
        reward_map[7][7] = 1

    return reward_map
```

این تابع در تکرار های مختلف اجرا میشود و نتایج مختلفی را بازمیگرداند

ابتدا ارزش تمام state ها صفر تعیین می شود.

State های دارای خوک های ملکه و دیوارها پاداش منفی می گیرند. تا زمانی که خوک ها وجود داشته باشند ابتدا نزدیک ترین خوک با استفاده از فاصله منهتن پیدا می شود و سپس ارزش 1+ دریافت میکند.

با تمام شدن خوک ها در صفحه state ترمینال و پایانی بازی ارزش 1+ دریافت میکند.

کاربرد تابع:

این تابع نقشه‌ای از مقادیر پاداش و جریمه ایجاد می‌کند که به الگوریتم value iteration کمک می‌کند:

1. هدایت عامل به سمت خوک‌ها یا اهداف دیگر: با تعیین پاداش مثبت برای خوک‌ها یا نقاط خاص، مسیر بهینه برای عامل مشخص می‌شود.
2. جریمه حرکت به سمت موانع یا نقاط ناخواسته: با اختصاص جریمه برای برخی موقعیت‌ها (مانند نقاط 'Q' و 'R')، عامل از حرکت در این مسیرها اجتناب می‌کند.
3. یادگیری بهتر در محیط بازی: این نقشه به عامل اجازه می‌دهد ارزش نسبی موقعیت‌های مختلف را یاد گرفته و حرکت خود را بر این اساس بهینه کند.

تابع value iteration

```python
def value_iteration(self, state, gamma=0.9, theta=1e-2):
    self.current_state = state
    self.reward_map = self.reward_function()
    self.transition_table = self.__calculate_transition_model(self.__grid_size, self.__probability_dict,
                                                              self.reward_map)
    grid = self.grid
    n_actions = 4
    grid_size = self.__grid_size
    transition_table = self.transition_table
    reward_map = self.reward_map

    V_history = []

    prev_V = np.zeros((grid_size, grid_size))
    policy = np.zeros((grid_size, grid_size, n_actions))

    while True:
        q = np.zeros((grid_size, grid_size, n_actions))

        for r in range(grid_size):
            for c in range(grid_size):
                state = (r, c)

                if grid[r][c] == 'R' or grid[r][c] == 'G':
                    continue

                for action in range(n_actions):
                    for prob, next_state, reward in transition_table[state][action]:
                        next_r, next_c = next_state
                        q[r, c, action] += (prob * (reward + (gamma * prev_V[next_r, next_c])))

        new_V = np.max(q, axis=2)

        delte = np.max(np.abs(new_V - prev_V))
        V_history.append(delte)
        if delte < theta:
            break

        prev_V = new_V.copy()

    policy = np.argmax(q, axis=2)
```

تابع value_iteration یک پیاده‌سازی از الگوریتم تکرار ارزش (Value Iteration) است که به‌منظور حل مسئله تصمیم‌گیری مارکوف (MDP) طراحی شده است. هدف این الگوریتم یافتن سیاست بهینه (Optimal Policy) در محیط است.

ایجاد Reward Map :

- ابتدا با استفاده از تابع reward_function، یک Reward Map برای محیط ایجاد می‌شود که به هر حالت مقدار پاداش یا جریمه اختصاص می‌دهد.

محاسبه مدل انتقالات:

- مدل انتقالات احتمال وقوع حالت‌های بعدی و پاداش متناظر هر انتقال را برای هر عمل محاسبه می‌کند.

اجرای الگوریتم تکرار ارزش:

- الگوریتم تکرار ارزش به‌صورت گام‌به‌گام اجرا می‌شود تا مقدار ارزش (Value) هر حالت محاسبه شود. این مقادیر نشان می‌دهند که هر حالت چقدر سودمند است و به عامل کمک می‌کنند تا بهترین عمل ممکن را در هر موقعیت انتخاب کند.

  نحوه کار:

  امید ریاضی به ازای اکشن های های مختلف در یک حالت محاسبه میشود و اکشنی که بیشترین امید ریاضی را داشته باشد به عنوان اکشن اصلی برای یک حالت انتخاب میشود.

همگرایی الگوریتم:

- الگوریتم تا زمانی که تغییرات مقادیر ارزش در طول تکرارها کمتر از یک مقدار آستانه مشخص شود، ادامه پیدا می‌کند. این روند تضمین می‌کند که نتایج نهایی دقیق و پایدار هستند.

استخراج سیاست بهینه:

- در پایان، بر اساس مقادیر ارزش محاسبه‌شده، سیاست بهینه برای هر حالت استخراج می‌شود. این سیاست مشخص می‌کند که عامل در هر حالت چه تصمیمی بگیرد تا بیشترین پاداش ممکن را کسب کند.

رسم نمودارهای v* , π^*, Convergence

```
plt.imshow(new_V, cmap='viridis', interpolation='none')
for i in range(new_V.shape[0]):
    for j in range(new_V.shape[1]):
        plt.text(j, i,  s: f'{new_V[i, j]:.2f}', ha='center', va='center', color='white')

plt.colorbar(label='Value')
plt.title("Value Table")
plt.show()



plt.plot(V_history)
plt.title("Convergence Table")
plt.show()


plt.imshow(policy, cmap='viridis', interpolation='none')
for i in range(policy.shape[0]):
    for j in range(policy.shape[1]):
        plt.text(j, i,  s: f'{policy[i, j]}', ha='center', va='center', color='white')

plt.colorbar(label='Value')
plt.title("Policy Table")
plt.show()
```

اجرای برنامه و تابع main :

```python
if __name__ == "__main__":

    mean_all_reward = []

    for i in range(3):
        print(f"environment: {i}")
        env = AngryBirds()
        state = env.reset()
        policy_0 = env.value_iteration(state, gamma=0.8, theta=1e-2)

        policy_list = []
        policy_list.append(policy_0)

        env_reward = []
        for j in range(5):
            state = env.reset()
            policy = policy_list[0]

            FPS = 20
            screen, clock = PygameInit.initialization()
            running = True

            grid = env.grid
            r, c = state
            full_reward = 0

            killed_pig = 0
            step_num = 0

            while running:

                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        running = False

                env.render(screen)

                if step_num > 50:
                    step_num = 0
                    policy = env.value_iteration(state, gamma=0.8, theta=1e-2)

                r, c = state
                action = policy[r][c]
                state, probability, reward_episode, done = env.step(action)
```

```
            step_num += 1
            if reward_episode == 250:
                step_num = 0
                killed_pig = killed_pig + 1
                if j == 0:
                    policy = env.value_iteration(state, gamma=0.8, theta=1e-2)
                    policy_list.append(policy)
                else:
                    policy = policy_list[killed_pig]

            full_reward += reward_episode
            if done:
                env_reward.append(full_reward)
                print(f"Episode finished with reward: {full_reward}")
                break

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

    mean_env_reward = np.mean(env_reward)
    print(f"Environment mean Reward:  {mean_env_reward}")

    mean_all_reward.append(mean_env_reward)

    pygame.quit()
mean_all_env_reward = np.mean(mean_all_reward)
print(f"All Environment mean Reward:  {mean_all_env_reward}")
```

در 3 محیط و در هر محیط 5 مرتبه اجرا میگیریم.

ابتدا تا حذف شدن تمام خوک ها، نزدیک خوک به عنوان هدف در نظر گرفته میشود و reward function و value iteration اجرا میشوند و سیاست بهینه برای رسیدن به آن خوک جستجو میشود.

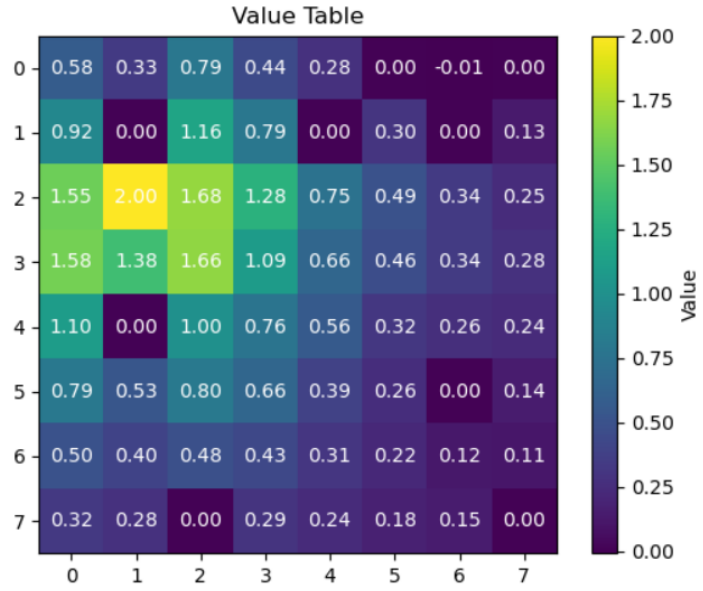
پس از رسیدن به یک خوک این عمل برای رسیدن به نزدیک ترین خوک دیگر اجرا میشود.

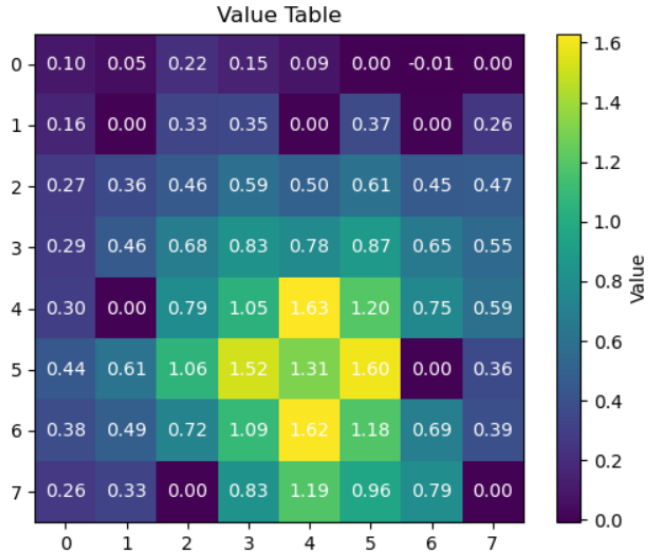
پس از اتمام تمام خوک ها در صفحه سیاست بهینه برای رسیدن به مقصد جستجو میشود.

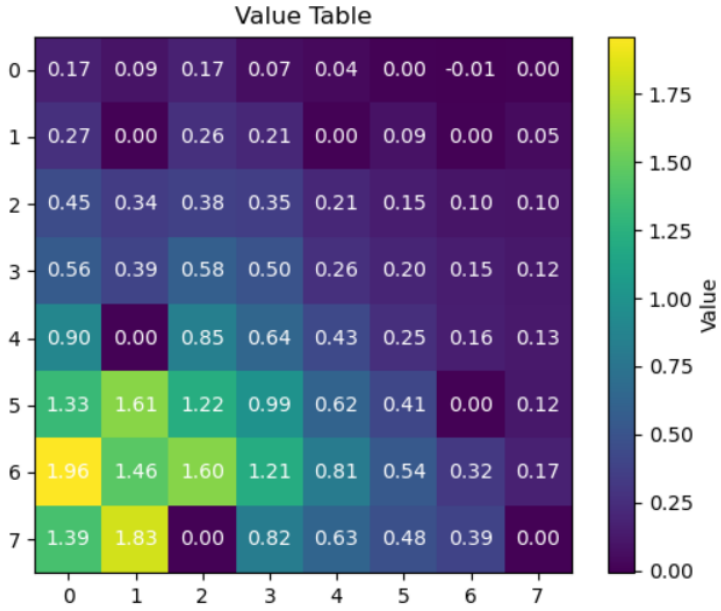
در صورت نرسیدن به هدف پس از 50 قدم سیاست جدیدی ایجاد می شود.

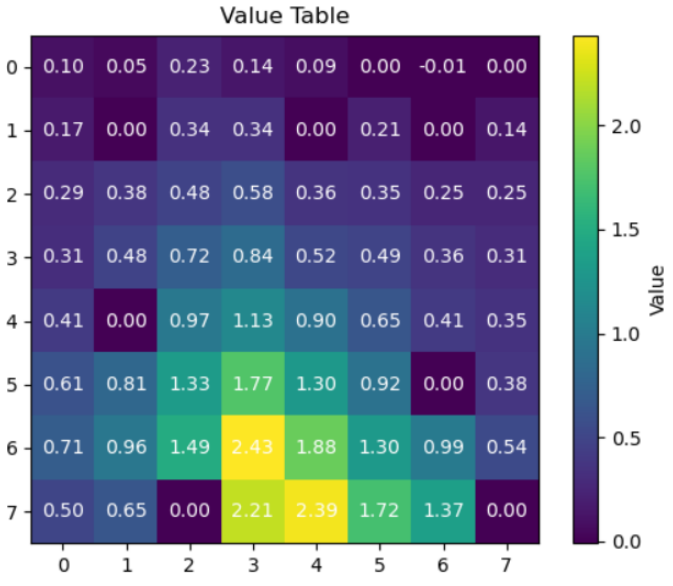
نتایج تست و اجرا :

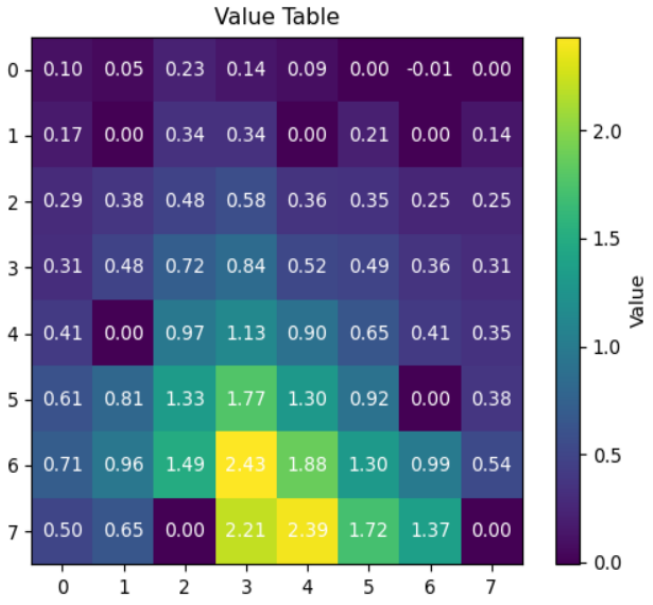
نمودار v^*

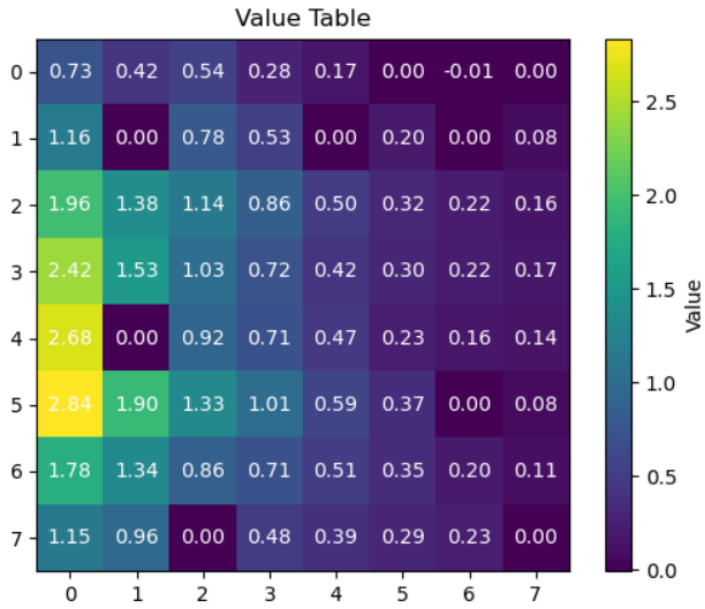


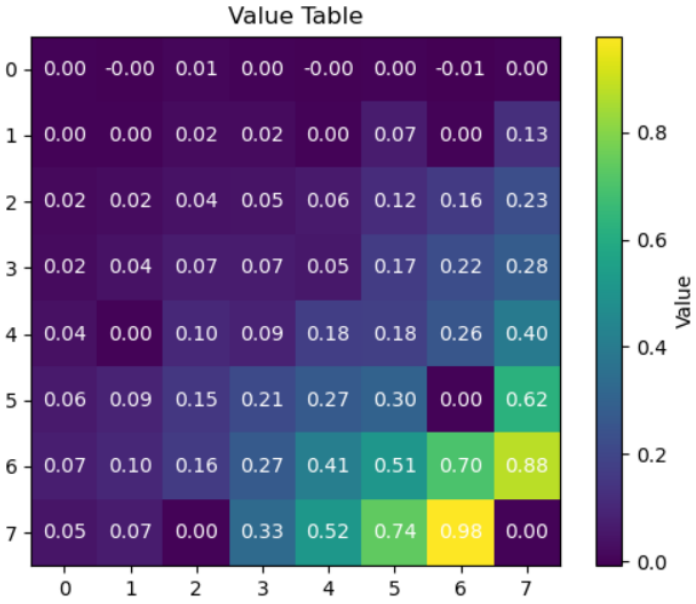




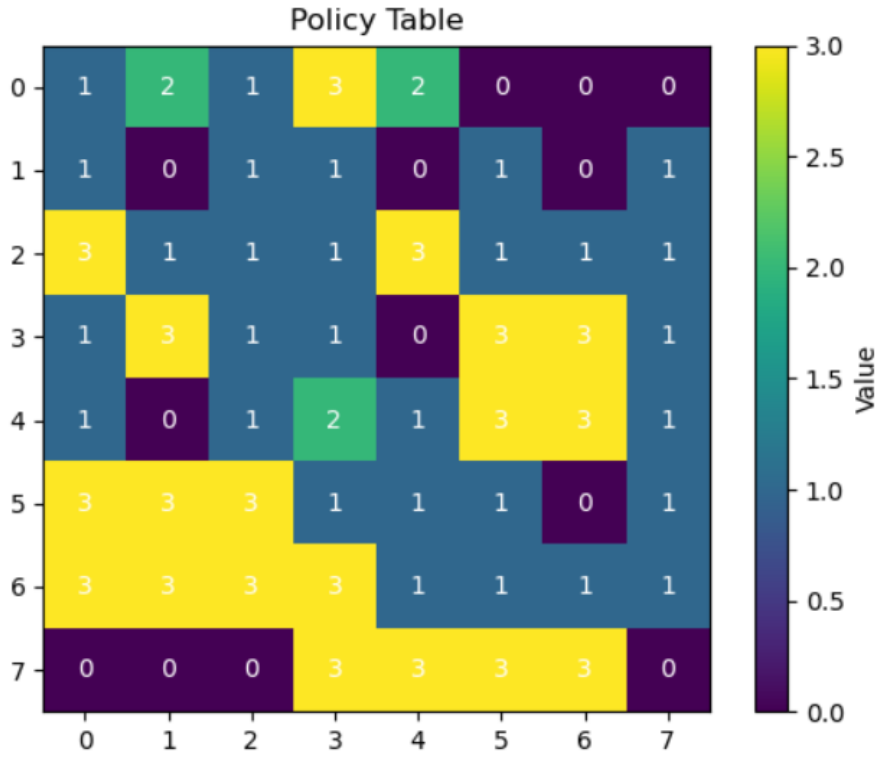



Value Table
0 0.10 0.05 0.23 0.14 0.09 0.00 -0.01 0.00
1 0.17 0.00 0.34 0.34 0.00 0.21 0.00 0.14
2 0.29 0.38 0.48 0.58 0.36 0.35 0.25 0.25
3 0.31 0.48 0.72 0.84 0.52 0.49 0.36 0.31
4 0.41 0.00 0.97 1.13 0.90 0.65 0.41 0.35
5 0.61 0.81 1.33 1.77 1.30 0.92 0.00 0.38
6 0.71 0.96 1.49 2.43 1.88 1.30 0.99 0.54
7 0.50 0.65 0.00 2.21 2.39 1.72 1.37 0.00
0 1 2 3 4 5 6 7
Value
0.0 0.5 1.0 1.5 2.0

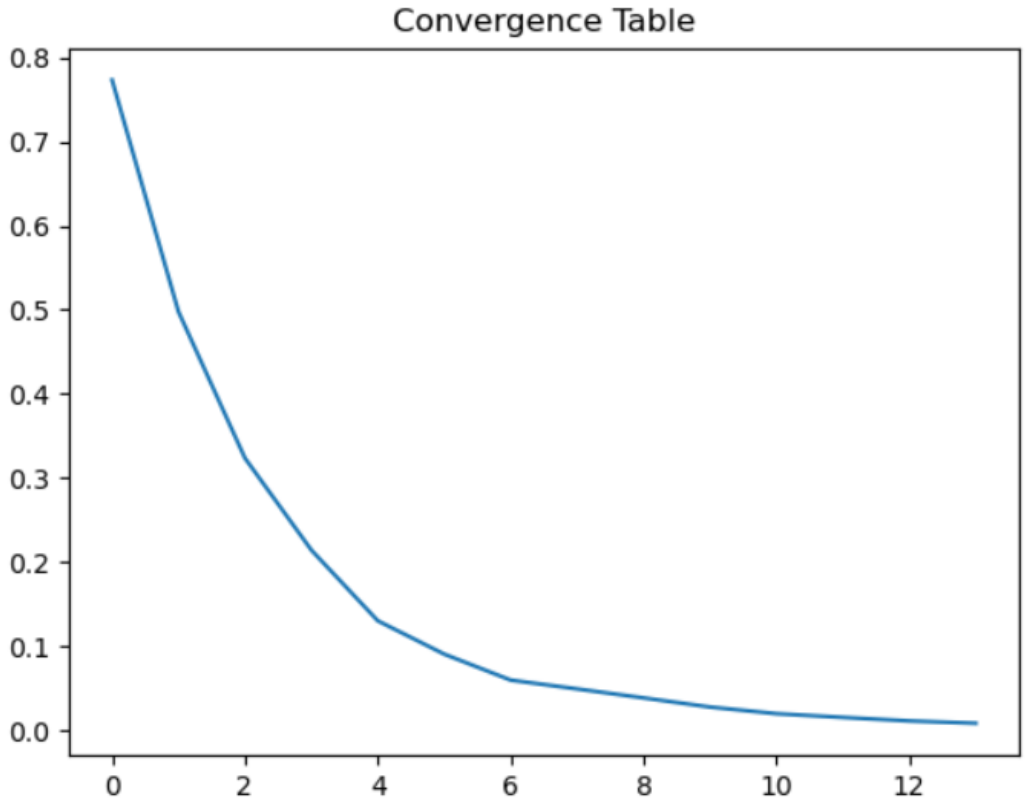

Value Table
0 0.73 0.42 0.54 0.28 0.17 0.00 -0.01 0.00
1 1.16 0.00 0.78 0.53 0.00 0.20 0.00 0.08
2 1.96 1.38 1.14 0.86 0.50 0.32 0.22 0.16
3 2.42 1.53 1.03 0.72 0.42 0.30 0.22 0.17
4 2.68 0.00 0.92 0.71 0.47 0.23 0.16 0.14
5 2.84 1.90 1.33 1.01 0.59 0.37 0.00 0.08
6 1.78 1.34 0.86 0.71 0.51 0.35 0.20 0.11
7 1.15 0.96 0.00 0.48 0.39 0.29 0.23 0.00
0 1 2 3 4 5 6 7
Value
0.0 0.5 1.0 1.5 2.0 2.5


Value Table
0 0.00 -0.00 0.01 0.00 -0.00 0.00 -0.01 0.00
1 0.00 0.00 0.02 0.02 0.00 0.07 0.00 0.13
2 0.02 0.02 0.04 0.05 0.06 0.12 0.16 0.23
3 0.02 0.04 0.07 0.07 0.05 0.17 0.22 0.28
4 0.04 0.00 0.10 0.09 0.18 0.18 0.26 0.40
5 0.06 0.09 0.15 0.21 0.27 0.30 0.00 0.62
6 0.07 0.10 0.16 0.27 0.41 0.51 0.70 0.88
7 0.05 0.07 0.00 0.33 0.52 0.74 0.98 0.00
0 1 2 3 4 5 6 7
Value
0.0 0.2 0.4 0.6 0.8

نمودار سیاست بهینه :

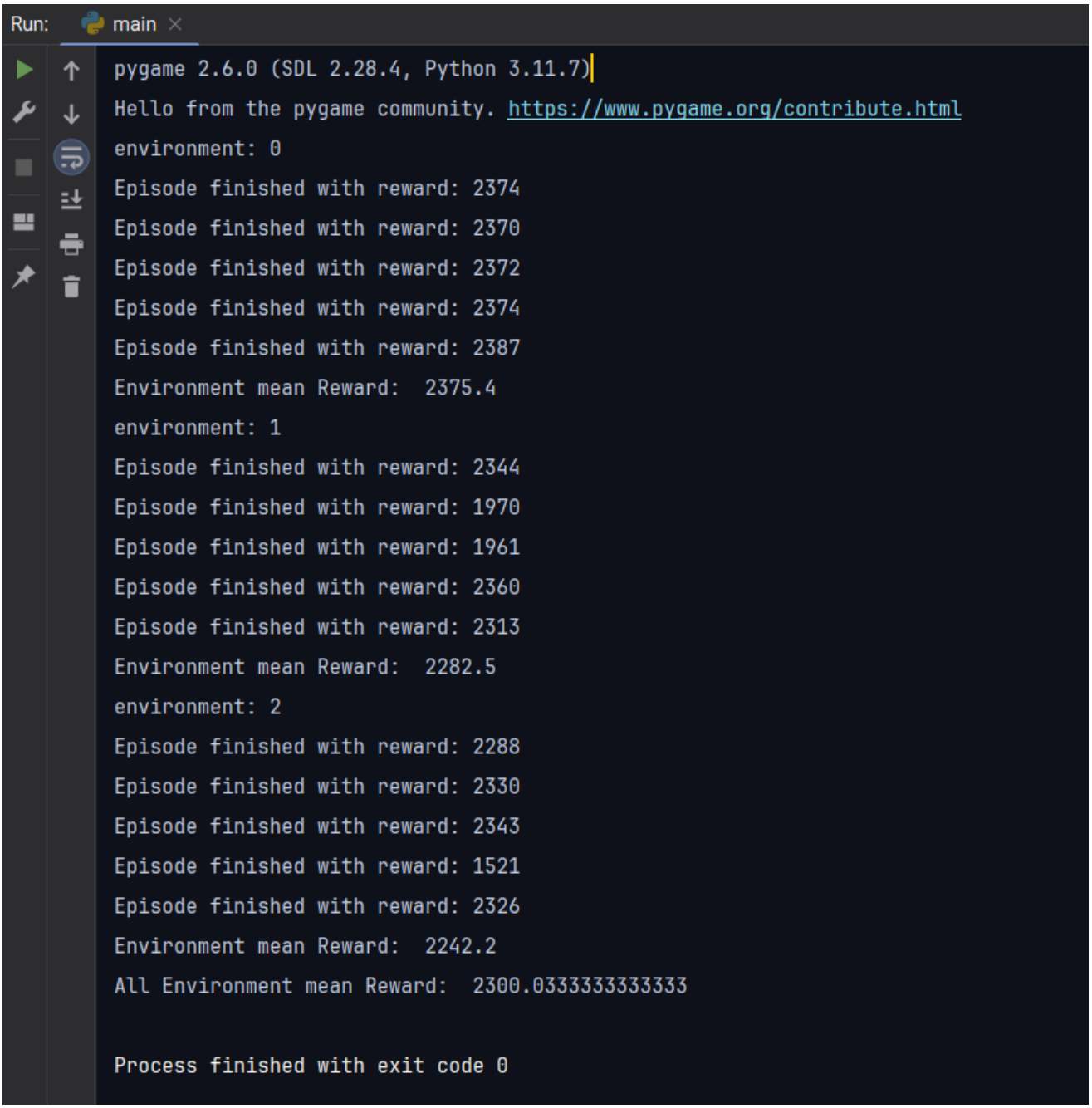


نمودار همگرایی :

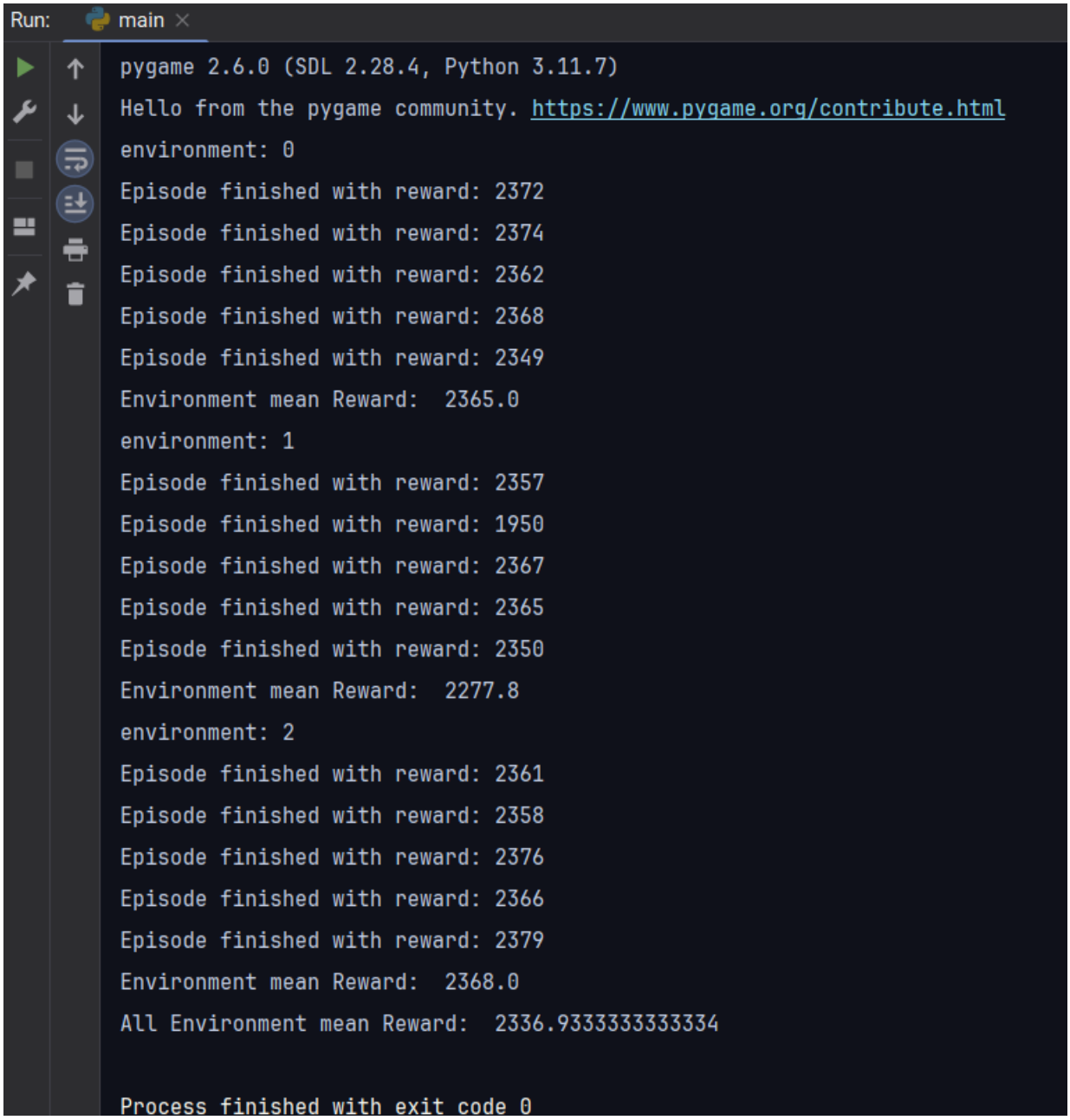


امتیازات کسب شده :

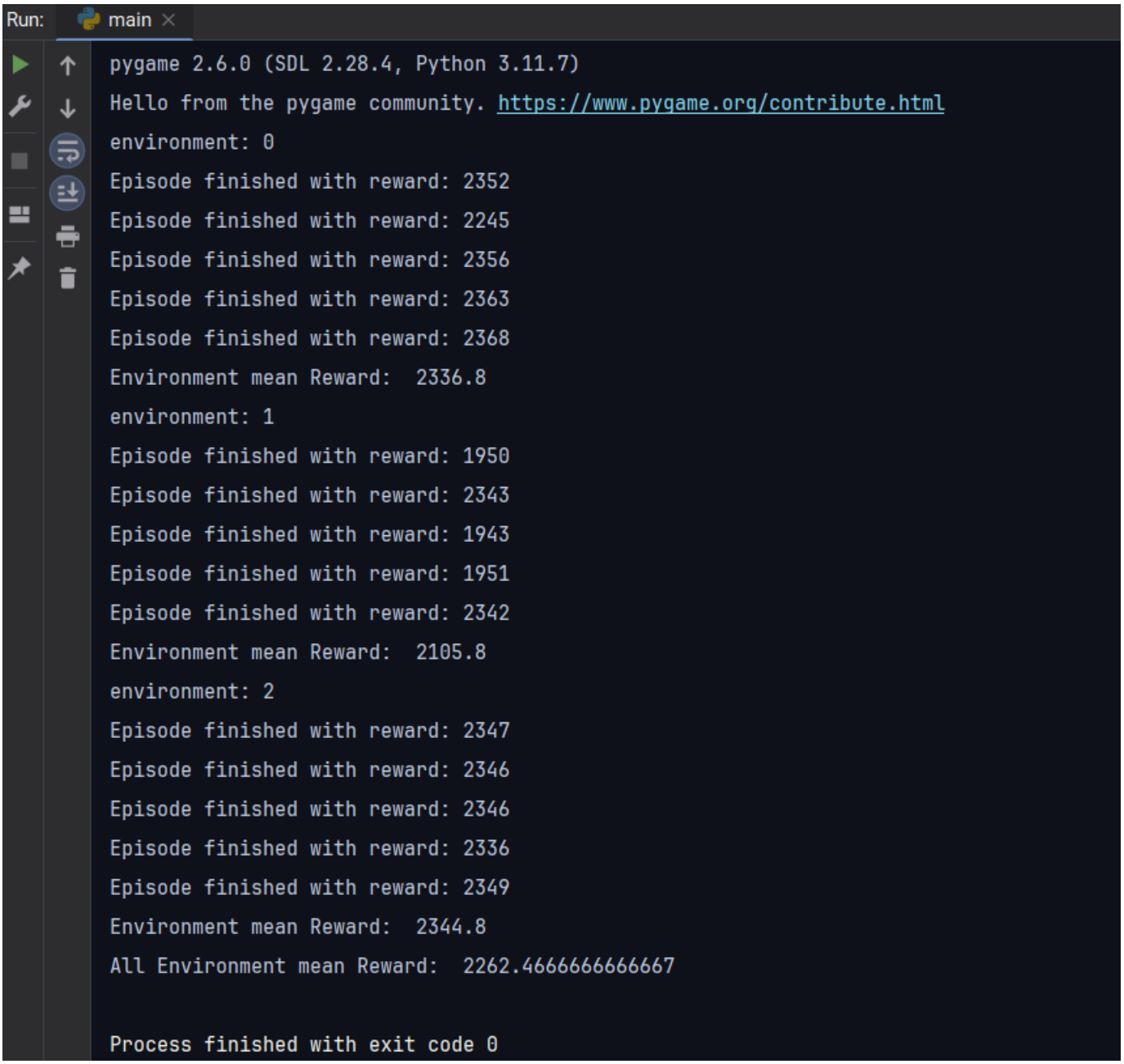
تست 1 :



تست 2 :


Run:
main
pygame 2.6.0 (SDL 2.28.4, Python 3.11.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
environment: 0
Episode finished with reward: 2372
Episode finished with reward: 2374
Episode finished with reward: 2362
Episode finished with reward: 2368
Episode finished with reward: 2349
Environment mean Reward: 2365.0
environment: 1
Episode finished with reward: 2357
Episode finished with reward: 1950
Episode finished with reward: 2367
Episode finished with reward: 2365
Episode finished with reward: 2350
Environment mean Reward: 2277.8
environment: 2
Episode finished with reward: 2361
Episode finished with reward: 2358
Episode finished with reward: 2376
Episode finished with reward: 2366
Episode finished with reward: 2379
Environment mean Reward: 2368.0
All Environment mean Reward: 2336.9333333333334
Process finished with exit code 0

تست 3 :


Run:
main
pygame 2.6.0 (SDL 2.28.4, Python 3.11.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
environment: 0
Episode finished with reward: 2352
Episode finished with reward: 2245
Episode finished with reward: 2356
Episode finished with reward: 2363
Episode finished with reward: 2368
Environment mean Reward: 2336.8
environment: 1
Episode finished with reward: 1950
Episode finished with reward: 2343
Episode finished with reward: 1943
Episode finished with reward: 1951
Episode finished with reward: 2342
Environment mean Reward: 2105.8
environment: 2
Episode finished with reward: 2347
Episode finished with reward: 2346
Episode finished with reward: 2346
Episode finished with reward: 2336
Episode finished with reward: 2349
Environment mean Reward: 2344.8
All Environment mean Reward: 2262.4666666666667
Process finished with exit code 0