Wumpus world with Q learning

کد نوشته شده، پیاده سازی الگوریتم یادگیری Q در یادگیری تقویتی هست، و صورت مسئله دنیای پامپوس است که در آن ربات باید از مقابله با هیولای وامپوس و چاه ها دوری کند و طلا را بدست بیاورد.

- محیط بازی ۵ در ۵ می باشد، دو چاه، یک وامپوس و یک طلا وجود دارد.
- با اجرای بازی، کاربر می تواند محل قرار گیری هر یک از آیتم ها را انتخاب کند.
- کاربر میتواند قبل از اجرای الگوریتم، نرخ یادگیری ، مقدار اپسیلون، گاما، تعداد بازی ها و اینکه آیا پارامتر های اپسیلون و نرخ یادگیری در طی آموزش کاهش داشته باشند را انتخاب کند.
- برای تنظیم پارامتر ها و اجرا، فایل readme.md رو مطالعه کنید.
- قسمت گرافیکی بازی توسط کتابخانه pygame نوشته شده است.
- لینک گیت هاب خودم. لینک

کلاس ها

برنامه دارای سه کلاس میباشد که برای راحتی نوشتن الگوریتم ، پیاده سازی شده است.

Board & Game

این دو کلاس حاوی اطلاعات و توابع بازی و محیط آن می باشند.
در کلاس board متغیر های موقعیت قرار گیری هر آیتم قرار گرفته است و توابعی از جمله ، رسم محیط و آیتم ها، برگرداندن حرکت های موجود، مشخص کردن پایان بازی و تعیین امتیاز هر حرکت را دارا می باشد.
کلاس game نیز متدهایی از جمله ریستارت و حرکت بعدی و پایان بازی را دارا میباشد.

Q_learning

در این کلاس ، الگوریتم یادگیری q پیاده سازی شده است. جدولی وجود دارد که تمامی احتمالات بازی در آن قرار گرفته است که در ابتدا مقادیر آنها صفر می باشد ولی با آموزش آن به مرور زمان، مقادیر بهینه برای بدست آوردن طلا در آن ذخیره میشود.
دو تابع دارد که یکی برای بدست آوردن حرکت بعدی می باشد و دیگری برای بهینه کردن مقادیر آن است.

تنظیمات

یک فایل نیز برای تنظیمات بازی قرار گرفته شده است که ابعاد بازی، رنگ ها و تصویر آیتم ها در آن قرار گرفته است.

الگوریتم یادگیری Q

بازی دارای ۵ سطر و ۵ ستون است و ربات قابلیت ۴ حرکت به بالا ، پایین، چپ و راست دارد. که با توجه به این نکات، هر محیط بازی حاوی ۱۰۰ حالت می باشد. پس ماتریس حالات ما برای الگوریتم ۵*۵*۴ میباشد.

برای بهینه سازی هر حالت، از الگوریتم زیر استفاده میکنیم.

```
def optimize(self, state, action, next_state, reward):
    old_value = self.q_table[state[0], state[1], action]
    next_max = np.max(self.q_table[next_state])

    new_value = (1 - self.alpha) * old_value + self.alpha * (reward + self.gamma * next_max)
    self.q_table[state[0], state[1], action] = new_value
```

در حین آموزش، دو حالت وجود دارد:

- حرکت رندوم
- حرکت توسط مقادیر Q

برای اینکه از بین این دو روش در هر نوبت یکی را انتخاب کنیم، از مقدار اپسیلون و عدد رندومی بین صفر و یک استفاده میکنیم.
اگر عدد رندوم کوچکتر از اپسیلون بود ، حرکت رندوم است و اگر بزرگتر باشد حرکت توسط جدول q می باشد.

```
if random.uniform(0, 1) < epsilon:
    action = random.randint(0, 3)
else:
    action = q_learner.select_action(state)
```

در این کد پس از هر ۱۰۰ بازی، مقادیر نرخ یادگیری و اپسیلون کاهش می یابد که می تواند توسط کاربر غیر فعال شود.

```
if epoch % 100 == 0:
    logging.info(f"Epoch: {epoch}, Penalties: {penalties}")
    if opt.decay:
        epsilon *= 0.9
        q_learner.alpha *= 0.9

        logging.info(f"Learning Rate : {q_learner.alpha} | Epsilon : {epsilon}")
```