

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق

درس یادگیری ماشین

استاد دکتر علیاری

سیدمحمدرضا حسینی

شماره دانشجویی: 40204584

گرایش: سیستم های الکترونیک دیجیتال

مینی پروژه شماره **4**

[**Google Colab**](https://colab.research.google.com/drive/1N-tLsvbrCP4qGxUmBW8WKAe26r1T2xVo?usp=sharing)

[**Github**](https://github.com/mohammadrezahosseini99/Machine-Learning.git)

**فهرست مطالب**

[سوال اول 1](#_Toc171740285)

[بخش آ 1](#_Toc171740286)

[بخش ب 19](#_Toc171740287)

[بخش ج 19](#_Toc171740288)

[بخش د 19](#_Toc171740289)

[بخش ه 19](#_Toc171740290)

# سوال اول

Wumpus به‌عنوان یک عنصر ثابت در نظر گرفته شده است اما سایر بخش های امتیازی، مانند توانایی شلیک توسط عامل و امتیاز مرتبط با کشتن Wumpus، لحاظ شده است. الگوریتم DQN هم به‌طور کامل بررسی و تمامی سوالات مربوط به آن پاسخ داده شده است.

## بخش آ

دنیای وومپوس یک مدل آموزشی کلاسیک در حوزه هوش مصنوعی است که توسط جان مک‌کارتی و ماروین مینسکی در دهه ۱۹۷۰ معرفی شد. این سناریو در یک محیط شبیه‌سازی شده ۴x۴ جریان دارد که اکتشافگر باید طلا را پیدا کرده و با موفقیت از محیط خارج شود. عناصر اصلی این محیط شامل اکتشافگر، طلا، وومپوس (موجود خطرناک)، حفره‌ها (موانع خطرناک) و خانه‌های خالی است. اکتشافگر توانایی حرکت به چهار جهت، تیراندازی برای کشتن وومپوس، برداشتن طلا و خروج از محیط را دارد. سیستم پاداش بر اساس کشف طلا و خروج ایمن یا شکست طراحی شده است.

برای حل این مسئله از روش‌های پیشرفته هوش مصنوعی مانند الگوریتم‌های جستجوی کلاسیک و یادگیری تقویتی (مانند Q-Learning و DQN) استفاده می‌شود. دنیای وومپوس به عنوان ابزار آموزشی، به درک اصول اساسی هوش مصنوعی و ارزیابی الگوریتم‌های نوین کمک می‌کند و به عنوان معیار آزمون و ارزیابی الگوریتم‌ها به کار می‌رود.

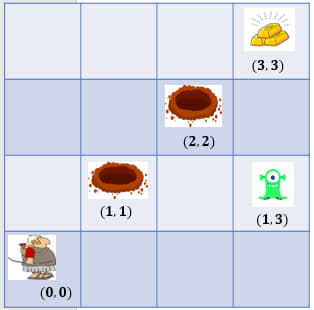
در ساخت محیط، عامل حداقل باید شش حرکت انجام دهد تا به طلا برسد و امتیازاتی بر اساس عملکرد خود دریافت کند. تنظیمات مختلف الگوریتم‌ها بر نتایج نهایی تاثیر می‌گذارد و هدف افزایش کارآمدی عملکرد عامل در محیط‌های دینامیک و چالشی است.

به دلیل پیچیدگی و مشکلات اجرای الگوریتم DQN، تصمیم گرفتیم مکان اشیاء را در طول آموزش ثابت نگه داریم تا عامل بهتر یاد بگیرد.

برای حل این مسئله با الگوریتم‌های یادگیری تقویتی به دستگاه‌های قدرتمند و زمان زیادی نیاز است. به دلیل محدودیت‌ها، نسخه ساده‌تری را انتخاب کردیم که در آن مکان اشیاء ثابت بوده و حرکت نمی‌کنند. با هر اجرا، عامل به نقطه شروع بازگشته و وامپوس نیز مجدداً زنده می‌شود.

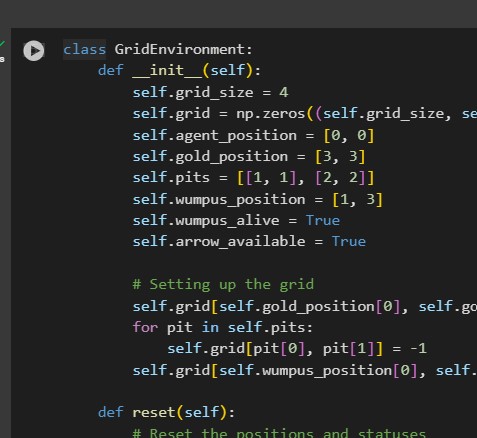
همان طور که گفتم مسئله Wumpus World یکی از نمونه‌های کلاسیک و تأثیرگذار در چالش‌های هوش مصنوعی است. این مسئله به‌طور دقیق نشان می‌دهد که چگونه می‌توان با استفاده از مجموعه‌ای از الگوریتم‌ها و تکنیک‌های مختلف، مشکلات پیچیده را حل کرد و عملکرد عامل‌ها را در محیط‌های دینامیک و چالش‌برانگیز بهبود بخشید. ارزیابی کدهای ارائه شده و توضیحات آن‌ها به درک عمیق‌تری از کاربرد این الگوریتم‌ها در دنیای Wumpus کمک می‌کند و امکان تحلیل و مقایسه نتایج حاصل از این پیاده‌سازی‌ها را فراهم می‌آورد.

در آغاز کار، با توجه به توضیح مسئله، به ساخت محیطی برای این چالش و تعریف قابلیت‌های لازم برای حرکت عامل در آن می‌پردازیم. محیط به صورت یک جدول چهار در چهار تعریف شده است که صفحه بازی ما را می‌سازد و دارای نقطه شروعی در مختصات (0,0) است. در این صفحه، علاوه بر عامل، اشیاء دیگری مانند طلا نیز وجود دارند که عامل در تلاش برای یافتن آن است و مکان آن توسط ما در نقطه (3,3) مشخص شده است که در شکل زیر قابل مشاهده است :

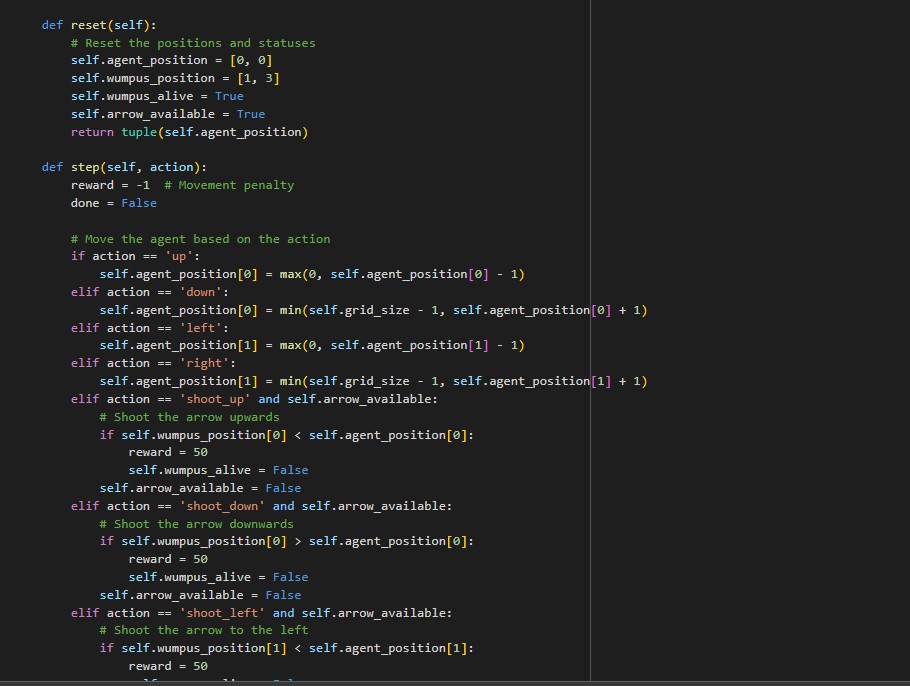


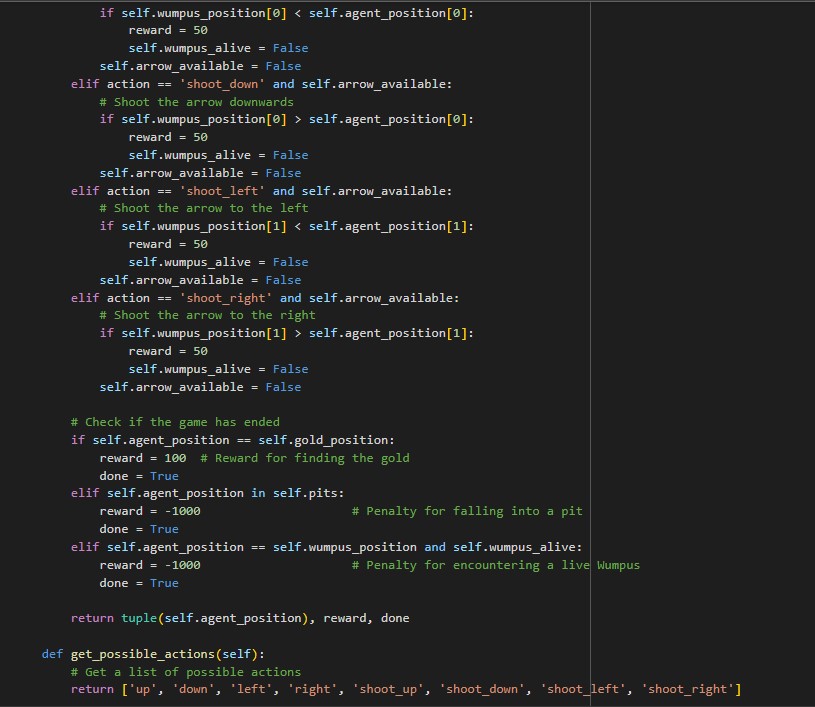
برای رسیدن به طلا، عامل باید حداقل شش حرکت انجام دهد. در ادامه، عامل برای هر حرکت که منجر به مرگ یا یافتن طلا نشود، امتیاز منفی (-1) دریافت می‌کند و در بهترین حالت، با کشتن Wumpus و دریافت جایزه +50 و همچنین یافتن طلا و دریافت جایزه +100، امکان رسیدن به مجموع امتیازات 145 وجود دارد. تغییرات در محیط و مکان اشیاء می‌تواند بر این حداکثر امتیاز تأثیر بگذارد. همچنین، تنظیم میزان Exploration الگوریتم به گونه‌ای که بتواند پس از مدتی به Exploitation بیشتری برسد، بر نتایج تأثیرگذار است و ممکن است عامل تنها به جای جستجوی حداکثر امتیاز، به یافتن طلا اکتفا کند که در این صورت حداکثر امتیاز قابل کسب +95 خواهد بود. این دلیل همگرایی الگوریتم Q-learning به امتیاز +95 است. در مراحل بعدی، با تغییر نحوه کاهش نرخ Exploration، این جنبه بیشتر مورد بررسی قرار می‌گیرد و انتظار می‌رود که با تنظیم مناسب، عامل بتواند به بیشترین امتیاز ممکن دست یابد، همانند آنچه در الگوریتم DQN رخ می‌دهد.

حال به سراغ کد نویسی و توضیح کد ها میرویم :



کد بالا یک کلاس به نام GridEnvironment ایجاد می‌کند که محیط شبیه‌سازی شده بازی Wumpus World را فراهم می‌کند. این محیط شامل یک شبکه ۴x۴ با خانه‌های خالی است و موقعیت‌های اولیه عناصر مختلف مانند عامل، طلا، چاه‌ها و وامپوس را تعیین می‌کند. همچنین، وضعیت اولیه وامپوس و تیر به True تنظیم می‌شوند.





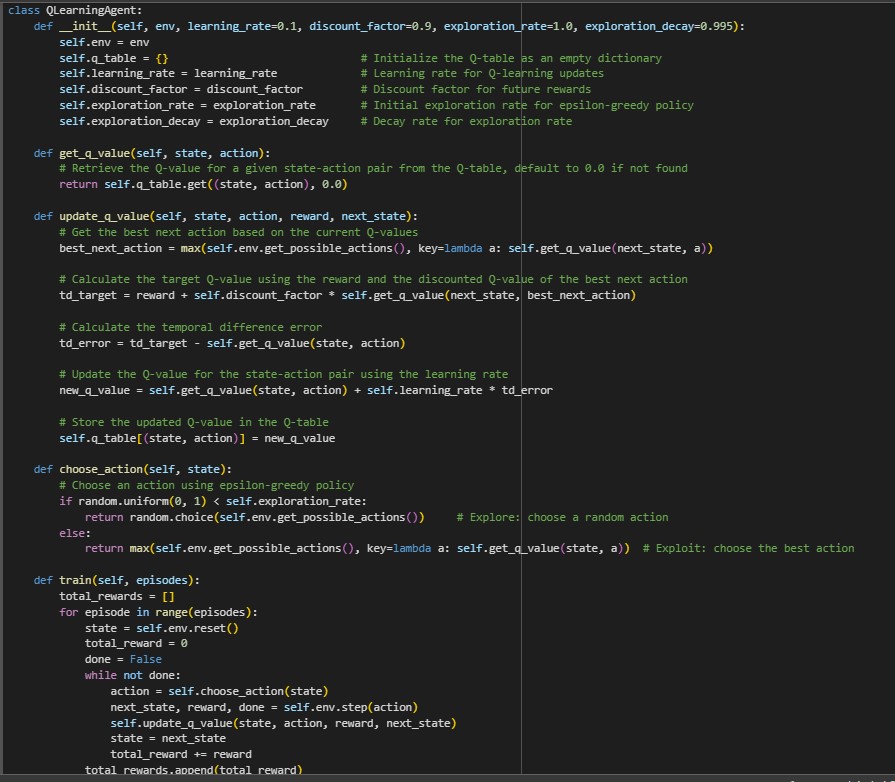
سه تابع در این کلاس به شبیه‌سازی محیط Wumpus World کمک می‌کنند:

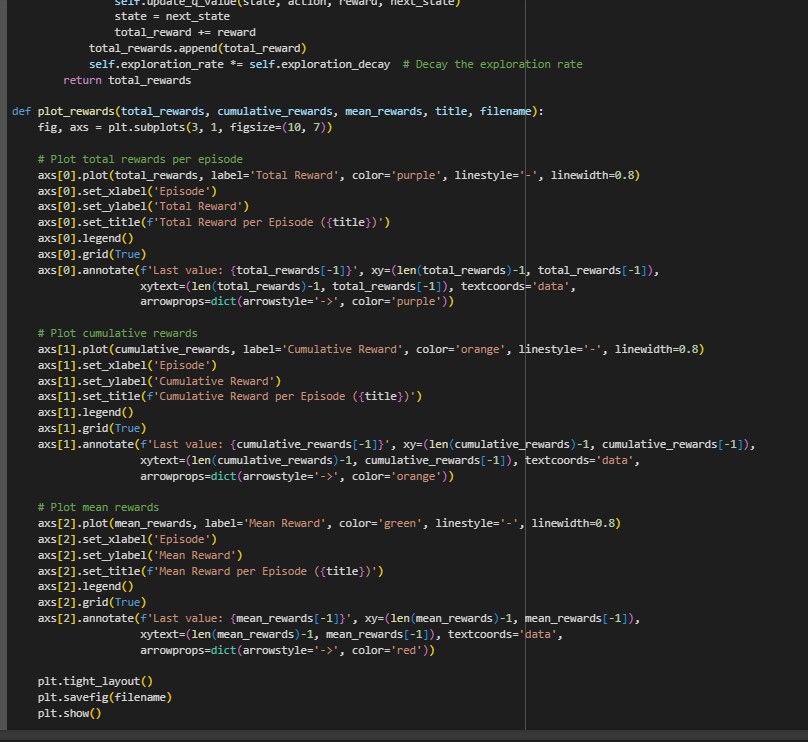
1. **تابع reset:** برای بازنشانی محیط به حالت اولیه، بازگرداندن عامل به نقطه شروع و تنظیم مجدد وضعیت وامپوس و تیر استفاده می‌شود.
2. **تابع step:** این تابع عملیات انجام شده در محیط را ارزیابی می‌کند. هر حرکت یک جریمه -۱ دارد. بر اساس عمل انتخاب شده (حرکت به جهات مختلف یا شلیک تیر)، موقعیت عامل تغییر می‌کند و وضعیت وامپوس و تیر به‌روزرسانی می‌شود. رسیدن به طلا ۱۰۰ امتیاز، افتادن در چاه یا برخورد با وامپوس زنده -۱۰۰۰ امتیاز، و کشتن وامپوس ۵۰ امتیاز دارد.
3. **تابع get\_possible\_actions:** این تابع لیستی از عملیات‌های ممکن را بازمی‌گرداند:

['up', 'down', 'left' 'right', 'shoot\_up', 'shoot\_down', 'shoot\_left', 'shoot\_right'].

این توابع برای مدیریت و ارزیابی عملکرد عامل در محیط Wumpus World طراحی شده‌اند و پایه‌ای برای پیاده‌سازی الگوریتم‌های یادگیری تقویتی هستند.

ا**لگوریتم Q-learning**





کلاس QLearningAgent:

- هدف: پیاده‌سازی الگوریتم Q-learning برای آموزش یک عامل در محیط‌های مختلف، از جمله Wumpus World.

توابع و ویژگی‌ها:

- init\_\_: این تابع سازنده پارامترهایی مانند محیط (env)، نرخ یادگیری (learning\_rate)، عامل تخفیف (discount\_factor)، نرخ اکتشاف اولیه (exploration\_rate) و نرخ کاهش اکتشاف (exploration\_decay) را می‌گیرد. جدول Q به عنوان یک دیکشنری خالی شروع می‌شود تا مقادیر Q جفت‌های حالت-عمل را ذخیره کند.

- get\_q\_value: مقدار Q مربوط به یک جفت حالت-عمل را بازمی‌گرداند. اگر جفت حالت-عمل در جدول Q موجود نباشد، مقدار پیش‌فرض ۰.۰ بازگردانده می‌شود.

- update\_q\_value: این متد برای به‌روزرسانی مقدار Q یک جفت حالت-عمل استفاده می‌شود. بهترین عمل برای حالت بعدی بر اساس مقادیر Q فعلی محاسبه و مقدار Q با استفاده از پاداش و مقدار Q تخفیف‌یافته به‌روزرسانی می‌شود.

- choose\_action: از سیاست ε-greedy استفاده می‌کند، به این معنا که با احتمال ε عمل تصادفی و با احتمال 1-ε بهترین عمل انتخاب می‌شود.

- train: عامل را از طریق تعدادی اپیزود آموزش می‌دهد. محیط در هر اپیزود بازنشانی می‌شود و عامل با انتخاب و اجرای عمل‌ها یاد می‌گیرد. مجموع پاداش‌ها در هر اپیزود جمع‌آوری شده و نرخ اکتشاف به مرور کاهش می‌یابد.

نمایش نتایج:

- تابع plot\_rewards: سه نمودار را برای نمایش عملکرد عامل ترسیم می‌کند:

- نمودار مجموع پاداش‌ها: پاداش کسب‌شده در هر اپیزود را نمایش می‌دهد.

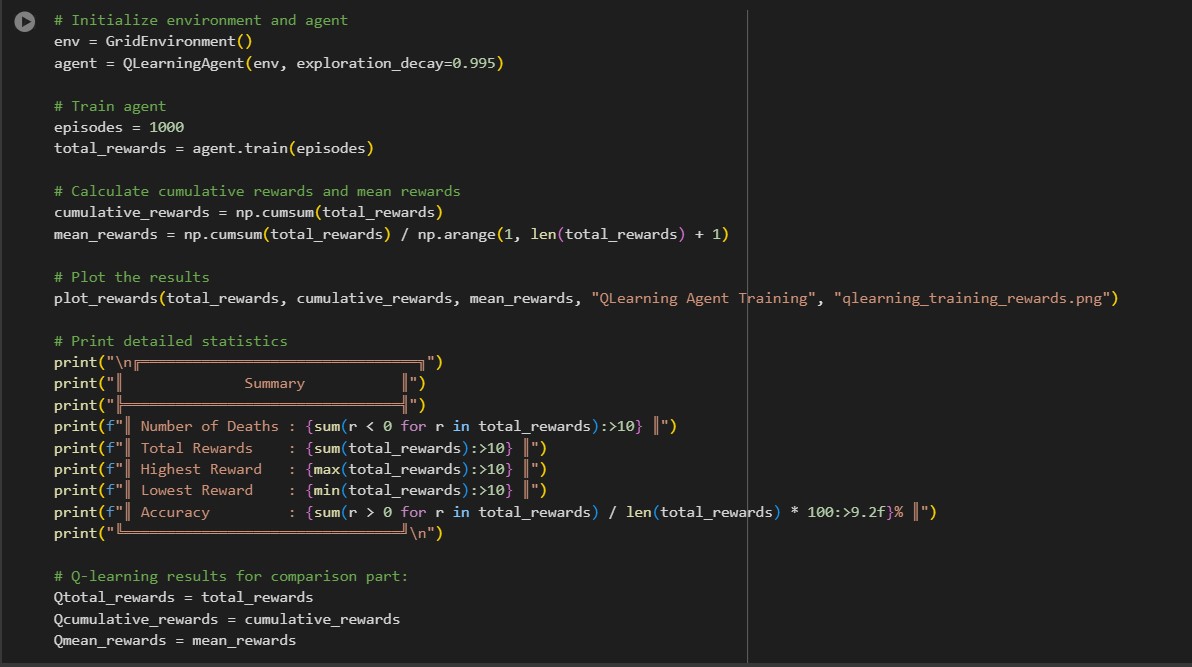
- نمودار پاداش‌های تجمعی: مجموع پاداش‌ها را در طول اپیزودها نشان می‌دهد.

- نمودار میانگین پاداش‌ها: میانگین پاداش‌ها را در طول اپیزودها ترسیم می‌کند.

نکات مهم:

- عامل با نرخ کاهش اکتشاف سریع‌تر (۰.۹۹۵) تنظیم شده تا به سرعت به سمت سیاست‌های بهینه همگرا شود.

- آموزش عامل برای ۱۰۰۰ اپیزود انجام می‌شود که می‌تواند تغییر کند.

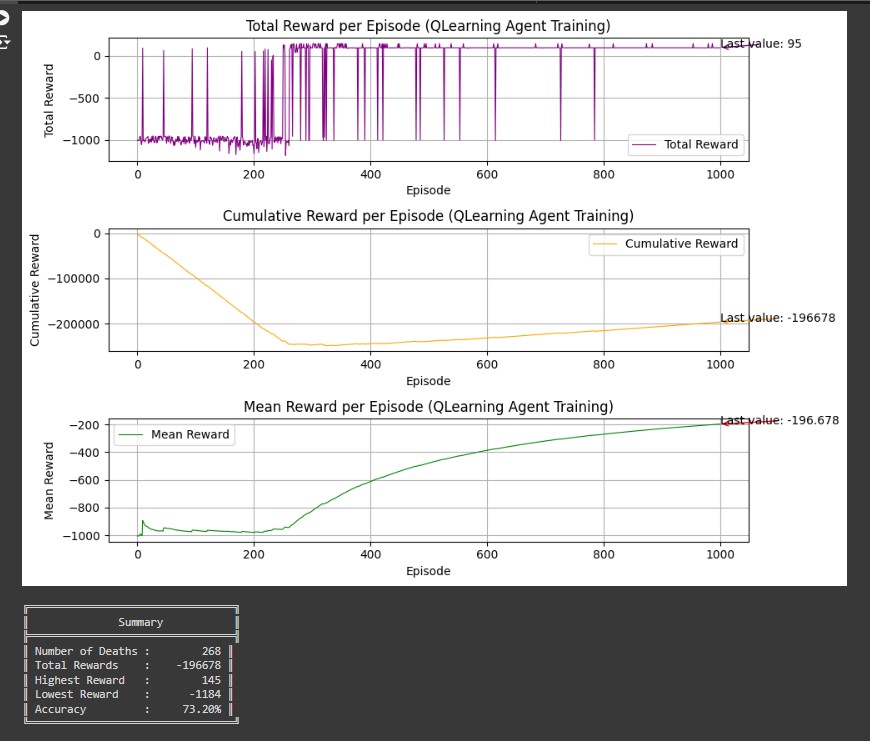


کد نهایی:

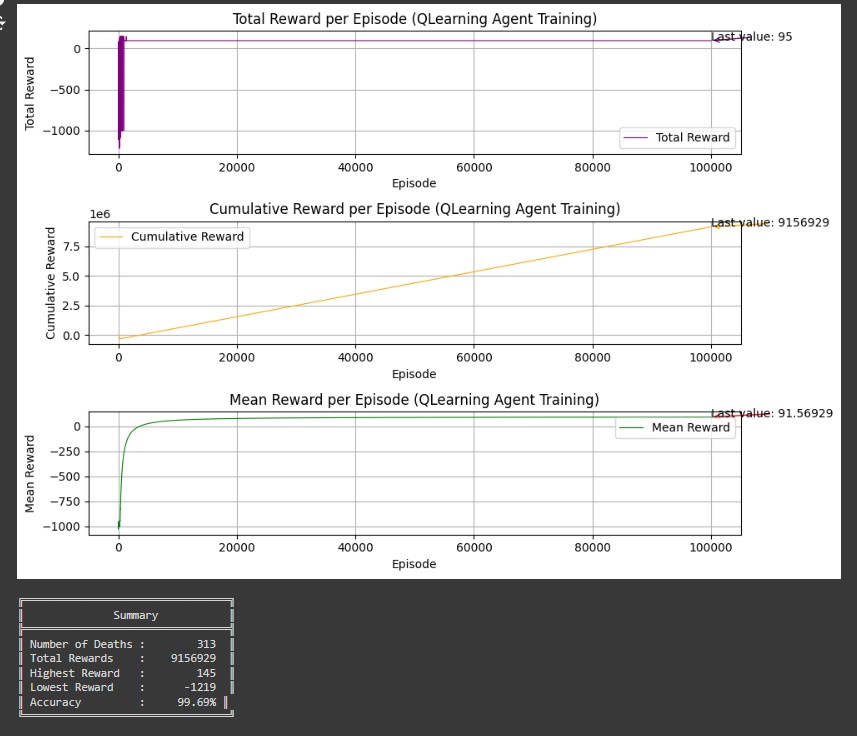
- ایجاد محیط GridEnvironment و عامل Q-learning.

- آموزش عامل و به‌روزرسانی مقادیر Q در طول اپیزودها.

- ذخیره نتایج پاداش‌ها در متغیر total\_rewards.



در بررسی انجام شده، از ۱۰۰۰ اپیزود، عامل 268 بار کشته شده و بالاترین امتیاز کسب‌شده ۱۴۵ بوده است. کمترین امتیاز به میزان منفی 1184 است. حدود 72 درصد اپیزودها با موفقیت و بدون کشته شدن به پایان رسیده‌اند. نمودار میانگین امتیازها روند صعودی دارد و هنوز به همگرایی نرسیده است، بنابراین قصد داریم الگوریتم را برای 100000 اپیزود نیز بررسی کنیم.



در این حالت مشاهده می شود که نمودار میانگین امتیازها تقریبا ثابت شده است و بنابراین همگرا شده است.

**الگوریتم DQN**

Deep Q-Networks (DQN) توسط محققان DeepMind برای غلبه بر محدودیت‌های Q-learning در محیط‌های پیچیده و بزرگ توسعه یافته است. برخلاف Q-learning که از جدول Q استفاده می‌کند، DQN از شبکه‌های عصبی عمیق برای ذخیره مقادیر Q-value بهره می‌برد، که این امکان را می‌دهد سیاست‌های پیچیده‌تری را یاد بگیرد.

**مزایا:**

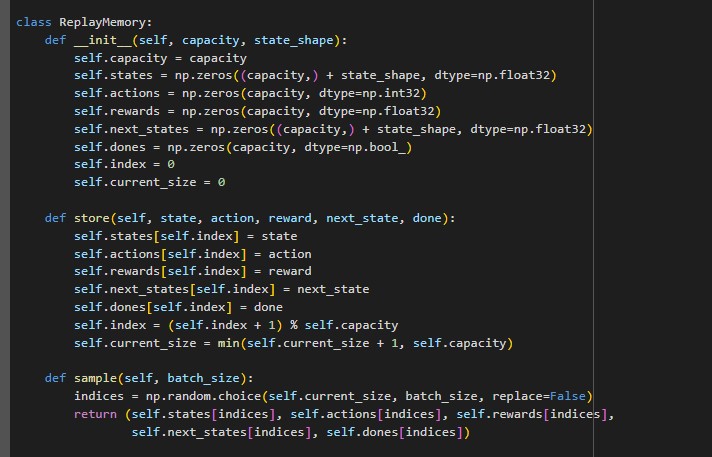
* عملکرد بهتر در محیط‌های با فضای حالت وسیع
* توانایی یادگیری توابع Q-value پیچیده و غیرخطی
* استفاده از تکنیک تجربه تکراری برای کاهش وابستگی به ترتیب تجربیات و بهبود کارایی یادگیری

**معایب:**

* نیاز به قدرت محاسباتی بالا
* پیچیدگی در تنظیم هایپرپارامترها
* عدم تضمین همگرایی مطمئن

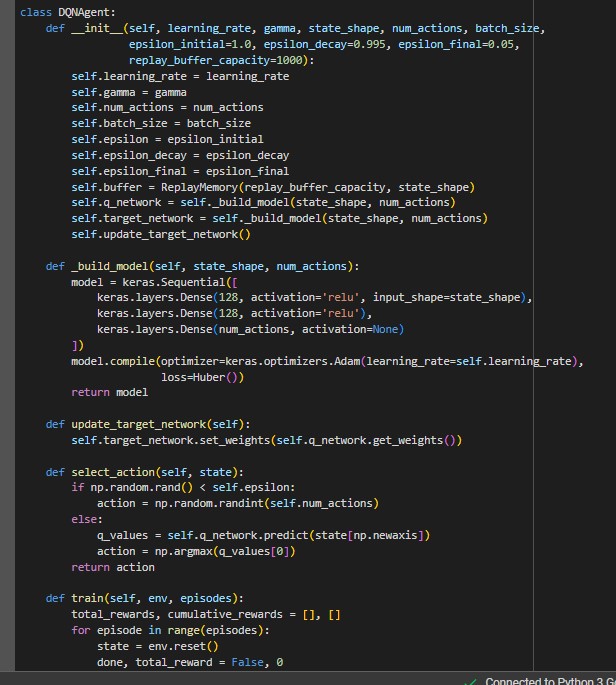
در مسئله Wumpus World، DQN برای تخمین مقادیر Q-value استفاده می‌شود. این الگوریتم با شبکه عصبی عمیق و تکنیک حافظه تجربه تکراری کار می‌کند. تجربیات عامل در حافظه ذخیره شده و به‌طور تصادفی برای به‌روزرسانی وزن‌های شبکه استفاده می‌شوند. حافظه بازپخش شامل وضعیت فعلی، عمل انجام شده، پاداش دریافتی، وضعیت بعدی و شاخص پایان اپیزود است و به پایداری و کارایی یادگیری کمک می‌کند. DQN به‌ویژه در بازی‌های ویدیویی و محیط‌های پیچیده کاربرد دارد.

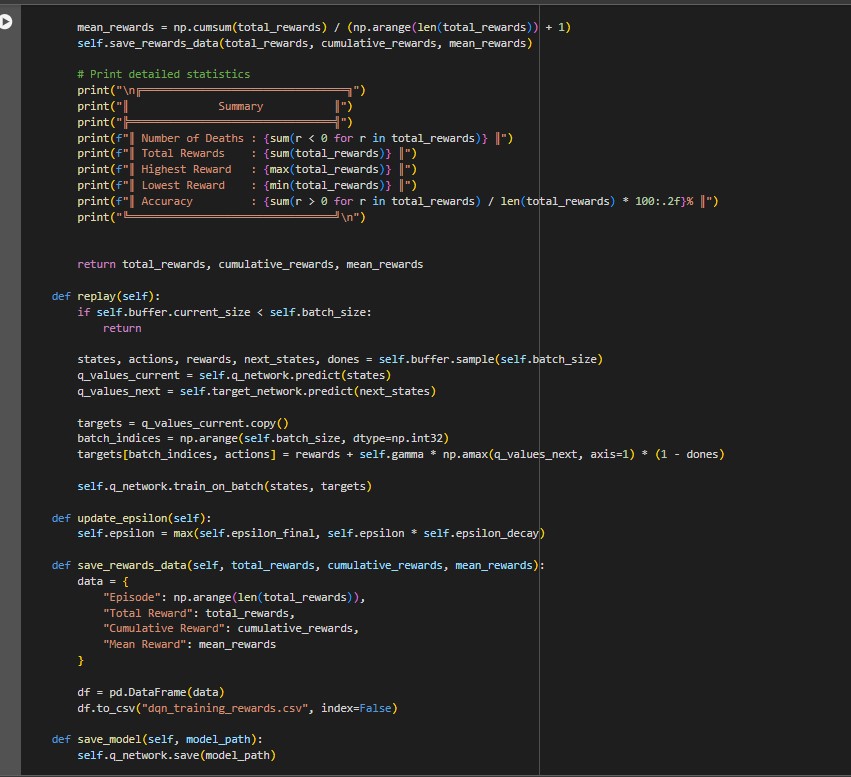
تعاریف اولیه مانند مجموعه‌های Reward Set و Action Set، ایجاد Environment، و قابلیت‌های Agent همانند قبل تعریف شده‌اند. در این بخش، تنها به بررسی بخش‌های جدید اضافه شده در این الگوریتم پرداخته می‌شود.

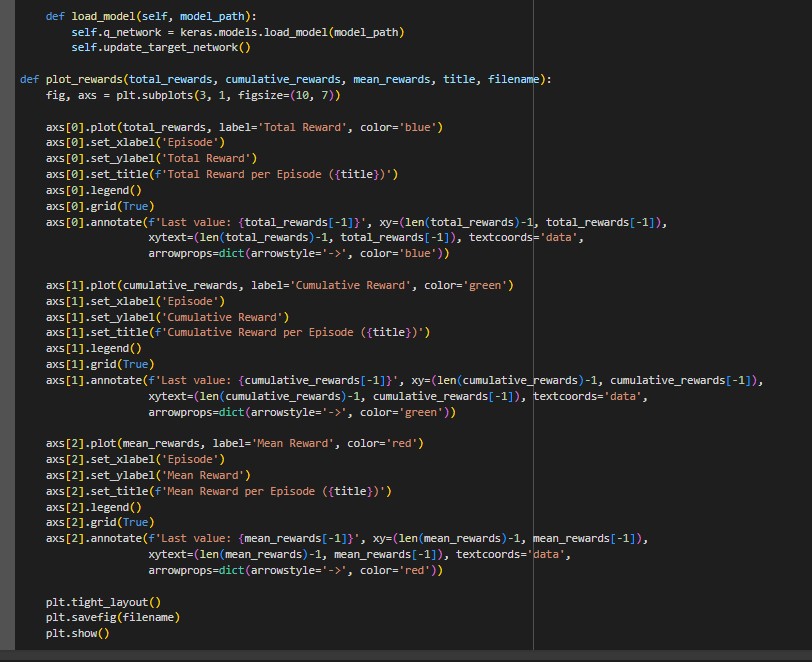


کلاس ReplayMemory برای ذخیره و مدیریت تجربیات عامل در الگوریتم‌های یادگیری تقویتی مانند DQN طراحی شده است. این کلاس با ذخیره تجربیات و انتخاب تصادفی آن‌ها برای یادگیری، پایداری و کارایی یادگیری سیاست‌های پیچیده را بهبود می‌بخشد.

* **\_\_init\_\_ سازنده:** ظرفیت حافظه و آرایه‌هایی برای ذخیره وضعیت‌ها، اعمال، پاداش‌ها، وضعیت‌های بعدی و شاخص پایان اپیزود را تعریف می‌کند.
* **store ذخیره تجربه:** تجربه جدید را ذخیره کرده و موقعیت فعلی و تعداد تجربیات را به‌روزرسانی می‌کند.
* **sample انتخاب تصادفی تجربیات:** یک دسته تصادفی از تجربیات را برای یادگیری انتخاب می‌کند.







کلاس DQNAgent برای پیاده‌سازی DQN استفاده می‌شود. پس از تعیین پارامترهای اولیه و شبکه عصبی، عامل در یک محیط آموزش داده می‌شود و نتایج آموزش ذخیره و رسم می‌شوند.

سازنده\_\_init\_\_ : تعریف و مقداردهی متغیرهای مورد نیاز برای یادگیری. ایجاد حافظه بازپخش و دو شبکه عصبی Q و هدف. به‌روزرسانی اولیه شبکه هدف.

متد \_build\_model: ساخت مدل شبکه عصبی با استفاده از Keras. تعریف لایه‌های شبکه و کامپایل آن.

متد update\_target\_network: به‌روزرسانی وزن‌های شبکه هدف با کپی کردن وزن‌های شبکه Q.

متد select\_action: انتخاب عمل بر اساس سیاست ϵ-greedy. انتخاب تصادفی عمل یا بهترین عمل بر اساس شبکه Q.

متد train: آموزش مدل با اجرای حلقه آموزشی برای تعداد مشخص اپیزود. ذخیره و به‌روزرسانی تجربیات در حافظه. فراخوانی متد replay برای به‌روزرسانی شبکه Q. به‌روزرسانی ϵ و شبکه هدف. ذخیره و رسم پاداش‌های آموزشی.

متد replay: به‌روزرسانی وزن‌های شبکه Q بر اساس تجربیات ذخیره شده. انتخاب تصادفی تجربیات از حافظه و آموزش شبکه Q با استفاده از داده‌های منتخب.

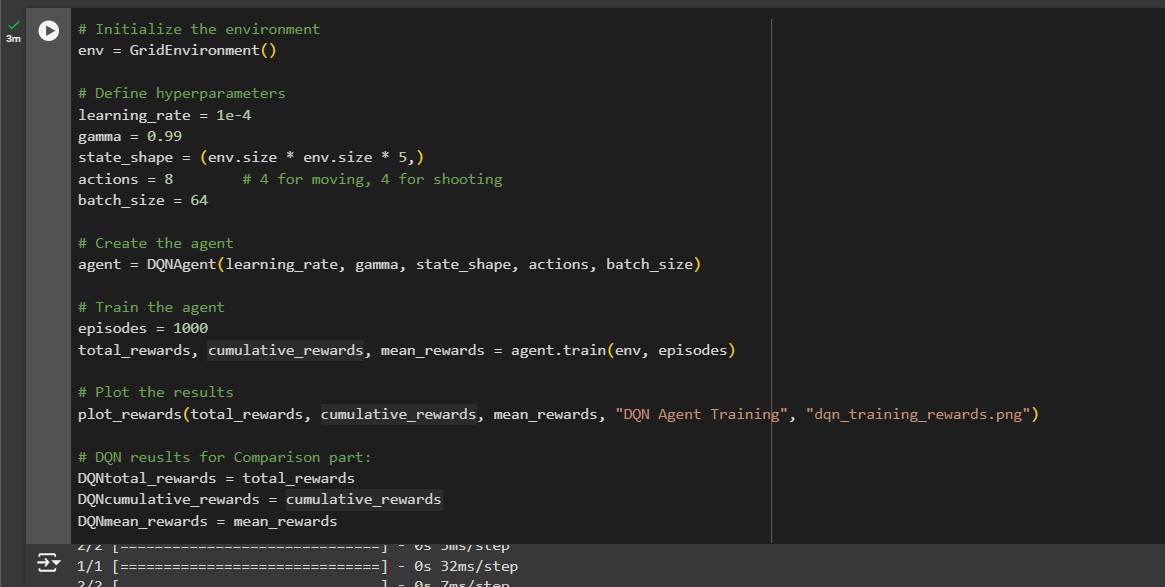
متد update\_epsilon: به‌روزرسانی مقدار ϵ با کاهش آن به نرخ کاهش مشخص تا مقدار نهایی.

متد save\_rewards\_data: ذخیره داده‌های پاداش آموزشی در فایل CSV.

متد save\_model: ذخیره مدل شبکه عصبی به صورت فایل.

متد load\_model: بارگذاری مدل ذخیره شده و به‌روزرسانی شبکه هدف.

و همچنین تابع plot\_rewards: رسم نمودار پاداش‌های کل، تجمعی و میانگین به ازای هر اپیزود. (این تابع خارج از کلاس است.)

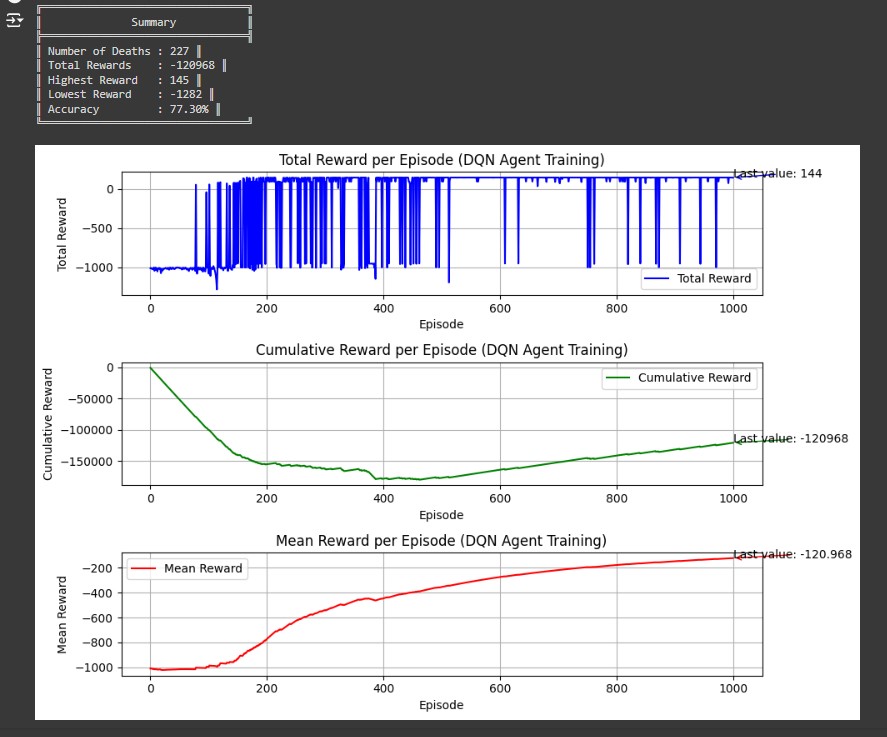


کد فوق با استفاده از یک محیط به نام GridEnvironment شروع می‌شود که به عنوان محیط آموزشی برای الگوریتم DQN عمل می‌کند. سپس، هایپرپارامترهای مختلفی اعمال می‌شوند که شامل نرخ یادگیری (learning\_rate)، ضریب کاهش پاداش (gamma)، شکل حالت (state\_shape)، تعداد اقدامات (actions) و اندازه بچ (batch\_size) می‌باشند.

یک عامل (agent) با استفاده از کلاس DQNAgent و هایپرپارامترهای مشخص شده ایجاد می‌شود. سپس، این عامل برای 1000 اپیزود آموزش داده می‌شود. در طی این اپیزودها، پاداش‌های کل (total rewards)، پاداش‌های تجمعی (cumulative rewards) و میانگین پاداش‌ها (mean rewards) برای هر اپیزود ذخیره و محاسبه می‌شوند.

در نهایت، نتایج آموزش به صورت نمودارها رسم می‌شوند تا بتوانیم عملکرد الگوریتم را به طور بصری بررسی کنیم. همچنین، پاداش‌های به دست آمده نیز برای مقایسه و تحلیل بیشتر ذخیره می‌شوند.

این فرآیند به طور خلاصه به منظور آموزش و ارزیابی عملکرد الگوریتم DQN با استفاده از یک محیط مصنوعی انجام می‌شود و نتایج نهایی از طریق نمودارها و آمارهای مربوطه مورد بررسی قرار می‌گیرند.



در طول آموزش به مدت 1000 اپیزود، عامل DQN حدود 227 بار کشته شده است که این عملکرد بهتری نسبت به الگوریتم Q-learning دارد.

مشابه الگوریتم Q-learning، عامل DQN به بهینه‌ترین سیاست دست یافته است و حداکثر امتیاز ممکن یعنی 145 را کسب کرده است.

در این آزمایش، در برخی از مواقع همگرایی در الگوریتم DQN حاصل نشده است. برای مثال، حتی بعد از 500 اپیزود، عامل همچنان پاداش‌های منفی دریافت کرده و کشته شده است. برای بررسی همگرایی، می‌توان این مدل را برای تعداد اپیزودهای بیشتری اجرا کرد.

علاوه بر این، مشخص شده است که عامل به طور قطع کشته نشده است و در بسیاری از موارد، در محیط به شدت فعال بوده است. اما، به دلیل حرکات مکرر و دریافت پاداش‌های منفی، نمی‌توان به وضوح تعیین کرد که آیا واقعاً کشته شده یا خیر. ایده‌آل است که نتایج نهایی هر اپیزود را به‌طور جداگانه بررسی کنیم تا بتوانیم به نتیجه‌ای قطعی‌تر درباره عملکرد عامل برسیم.

به دلیل زمان طولانی اجرای الگوریتم و محدودیت‌های زمانی، به این نتایج بسنده می‌کنیم. با این حال، الگوریتم DQN همچنان دارای پتانسیل بهبود است، به‌ویژه با کاهش نرخ کاوش و تنظیم نرخ یادگیری برای جلوگیری از گیر افتادن در مینیمم‌های محلی و تعیین تعداد حرکات معقول در هر اپیزود.

در مجموع، نتایج نشان می‌دهد که الگوریتم DQN نسبت به Q-learning عملکرد بهتری داشته و سریع‌تر به صعود در امتیازها دست یافته است، اگرچه همگرایی آن کار آسانی نیست و نیازمند بررسی و تنظیمات دقیق‌تری است و زمان اجرای بیشتری دارد.

## بخش ب

## بخش ج

## بخش د

## بخش ه