

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/1SMzivYSNgZK_Hhtvh1B_VXfgiINEEegz?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 21](#_Toc171740288)

[بخش د 24](#_Toc171740289)

[بخش ه 25](#_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.

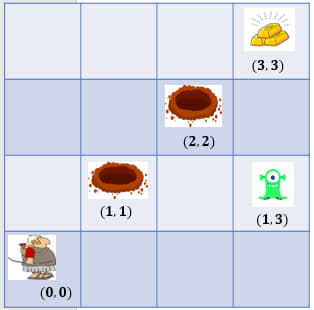
در ساخت محیط، عامل حداقل باید شش حرکت انجام دهد تا به طلا برسد و امتیازاتی بر اساس عملکرد خود دریافت کند. تنظیمات مختلف الگوریتم‌ها بر نتایج نهایی تاثیر می‌گذارد و هدف افزایش کارآمدی عملکرد عامل در محیط‌های دینامیک و چالشی است.

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.

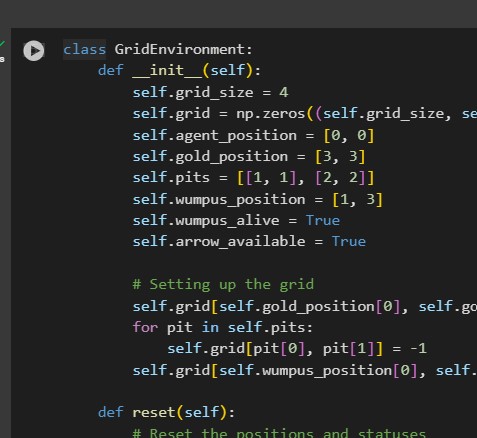
در آغاز کار، با توجه به توضیح مسئله، به ساخت محیطی برای این چالش و تعریف قابلیت‌های لازم برای حرکت عامل در آن می‌پردازیم. محیط به صورت یک جدول چهار در چهار تعریف شده است که صفحه بازی ما را می‌سازد و دارای نقطه شروعی در مختصات (0,0) است. در این صفحه، علاوه بر عامل، اشیاء دیگری مانند طلا نیز وجود دارند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



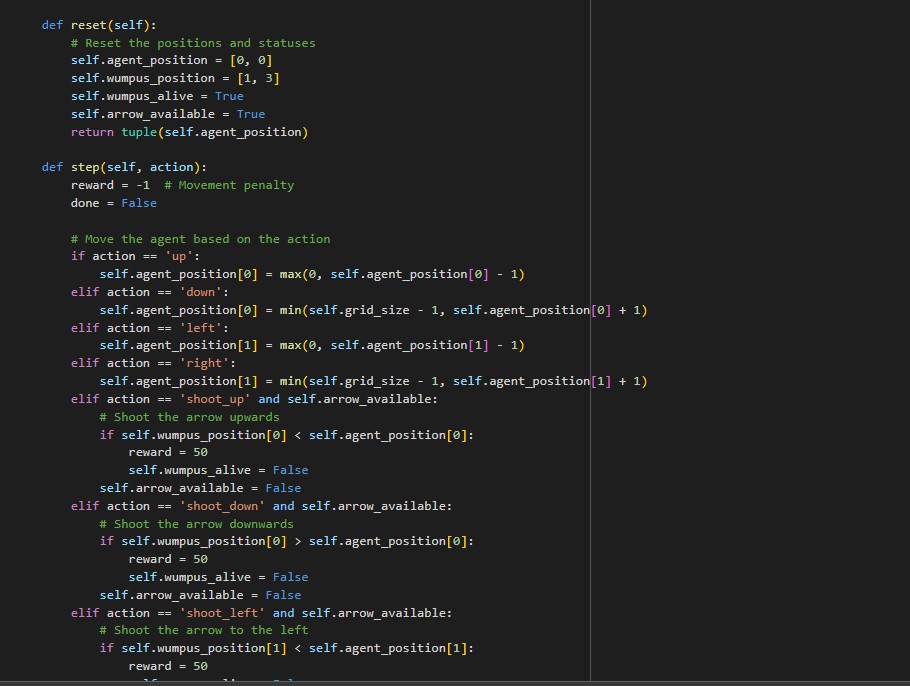
برای رسیدن به طلا، عامل باید حداقل شش حرکت انجام دهد. در ادامه، عامل برای هر حرکت که منجر به مرگ یا یافتن طلا نشود، امتیاز منفی (-1) دریافت می‌کند و در بهترین حالت، با کشتن Wumpus و دریافت جایزه +50 و همچنین یافتن طلا و دریافت جایزه +100، امکان رسیدن به مجموع امتیازات 145 وجود دارد. تغییرات در محیط و 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.

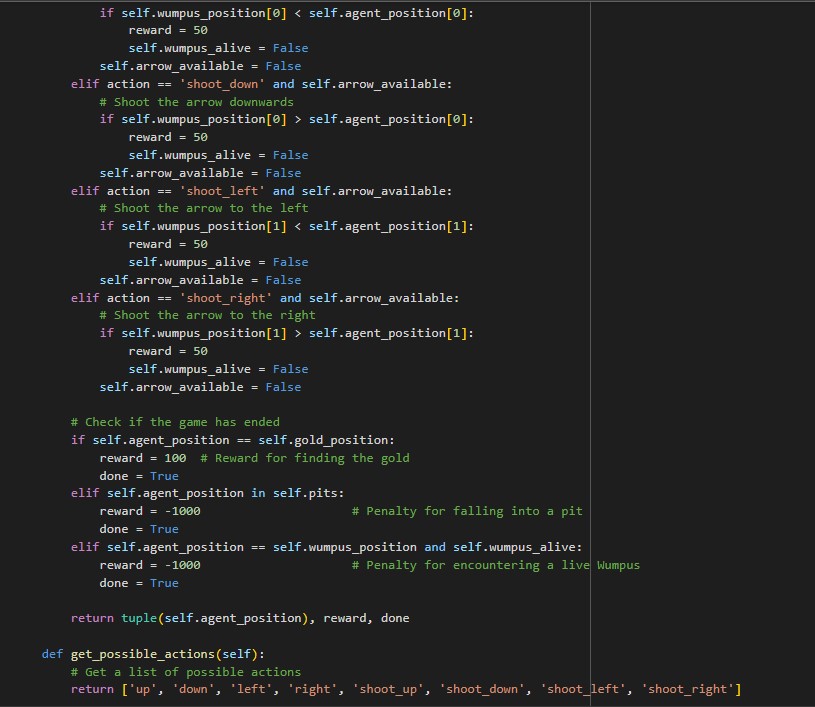
موارد امتیازی هم لحاظ شدند.

حال به سراغ کد نویسی و توضیح کد ها میرویم : ( پیاده سازی و شرح نحوه تعریف محیط و قابلیت های عامل)



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:

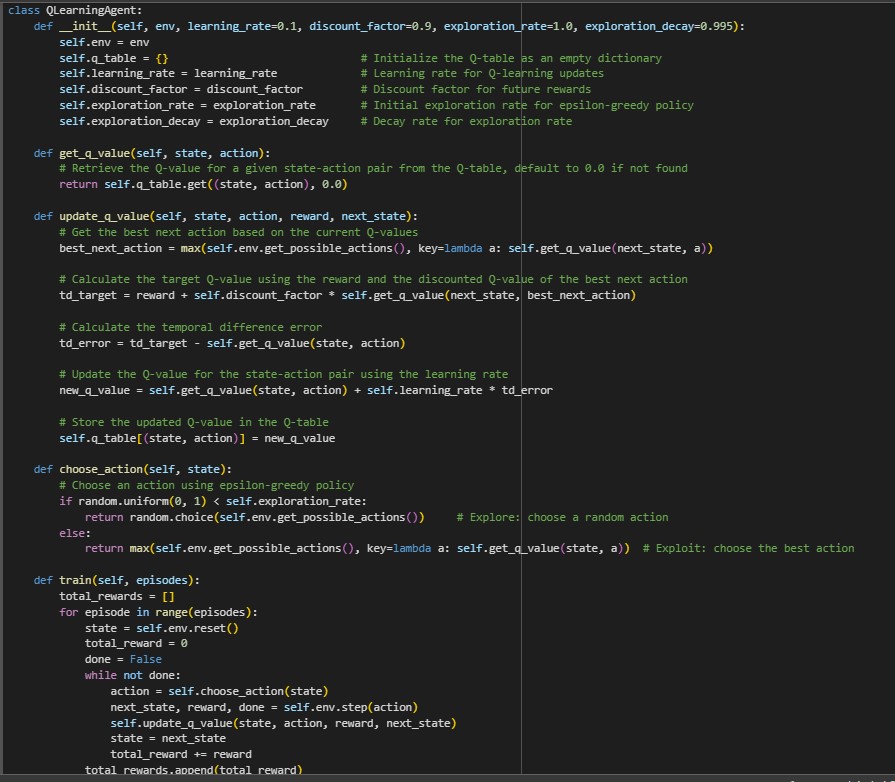
['up', 'down', 'left' 'right', 'shoot\_up', 'shoot\_down', 'shoot\_left', 'shoot\_right'].

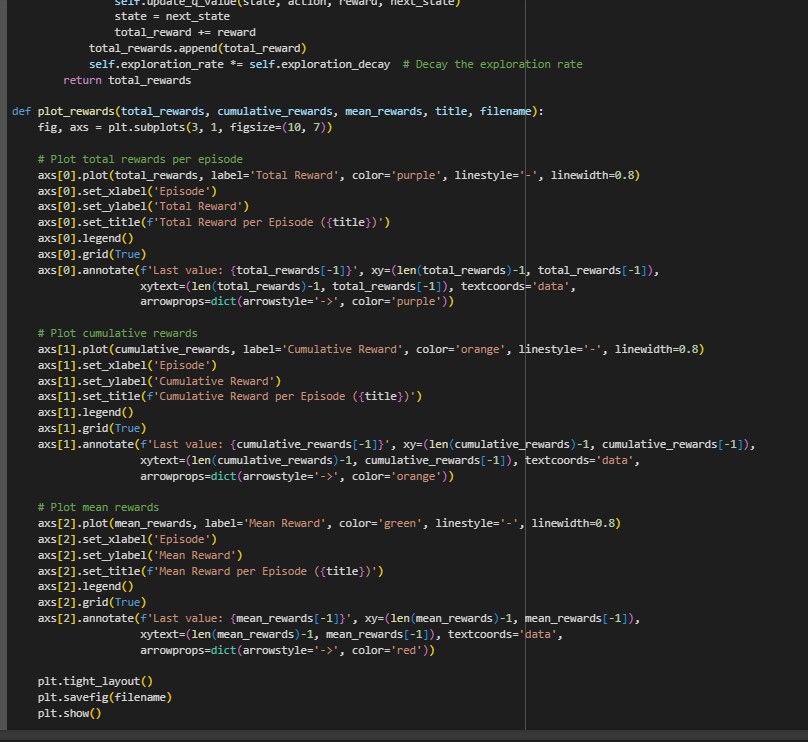
این توابع برای مدیریت و ارزیابی عملکرد عامل در محیط Wumpus World طراحی شده‌اند و پایه‌ای برای پیاده‌سازی الگوریتم‌های یادگیری تقویتی هستند.

دو الگوریتم را بررسی میکنیم. Q learning و DQN که در ادامه به آن ها می پردازیم.

ا**لگوریتم Q-learning**

روش Q-learning یکی از تکنیک‌های یادگیری تقویتی است که به‌طور گسترده برای حل مسائل پیچیده در حوزه هوش مصنوعی استفاده می‌شود. این روش به‌ویژه در مسائلی کاربرد دارد که در آن‌ها یک عامل باید از طریق تعامل با محیط، استراتژی بهینه‌ای برای رسیدن به هدف بیابد . Q-learning یک روش یادگیری بدون مدل است که به عامل امکان می‌دهد بدون داشتن اطلاعات قبلی از محیط، یک سیاست برای رسیدن به هدف یاد بگیرد. این روش بر اساس یادگیری مقادیر عمل-وضعیت (state-action values) یا Q-values عمل می‌کند. Q-value نشان‌دهنده امتیاز پیش‌بینی‌شده‌ای است که عامل می‌تواند با انجام یک عمل خاص در یک وضعیت خاص انتظار داشته باشد. هدف اصلی در Q-learning، به‌روزرسانی مقدار Q برای هر جفت وضعیت-عمل است تا Q-value به مقدار واقعی نزدیک شود. کلاس Qlearning را به صورت زیر تعریف می کنیم.





کلاس QLearningAgent را تعریف می کنیم که شامل متد ها و اتربیوت هایی است که به اختصار توضیح می دهم.

- 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.

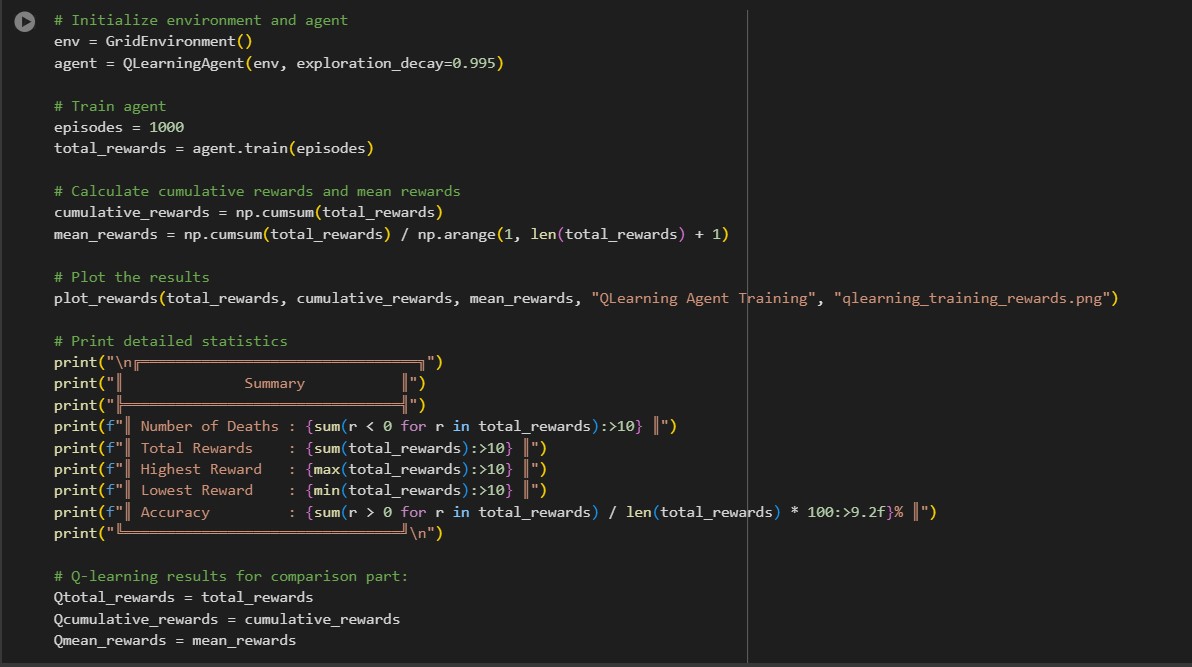
- نمودار میانگین پاداش‌ها: میانگین پاداش‌ها را در طول اپیزودها ترسیم می‌کند.

نکات :

- 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 اما بنا به خواسته صورت مسئله این مقدار فرض کردم.

حال مقدار دهی اولیه را انجام می دهیم :



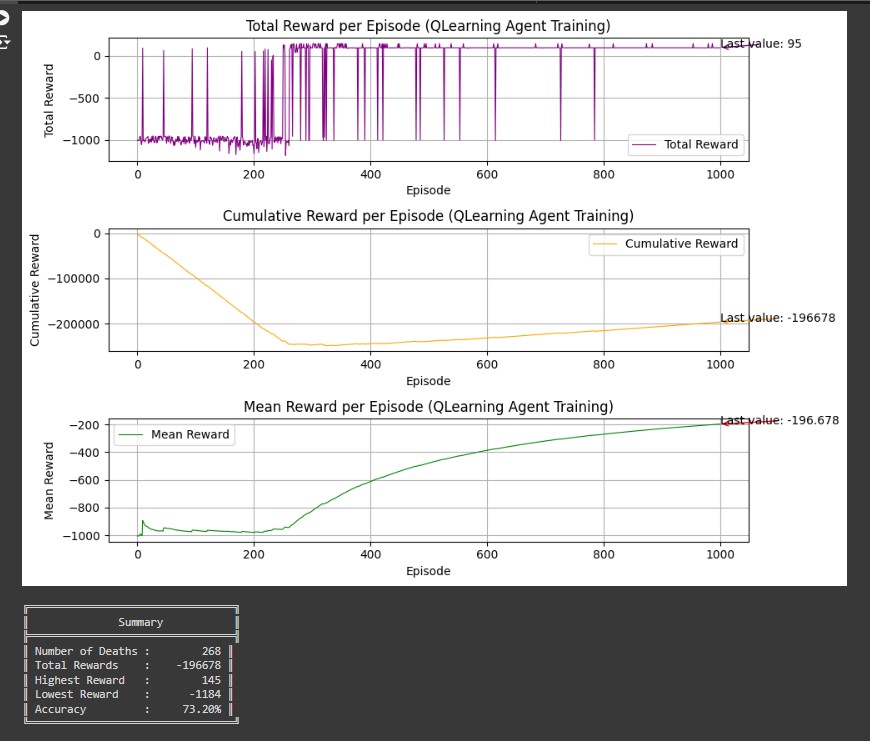
گام های زیر به طور خلاصه طی شده است :

- ایجاد محیط GridEnvironment و عامل Q-learning.

- آموزش عامل و به‌روزرسانی مقادیر Q در طول اپیزودها.

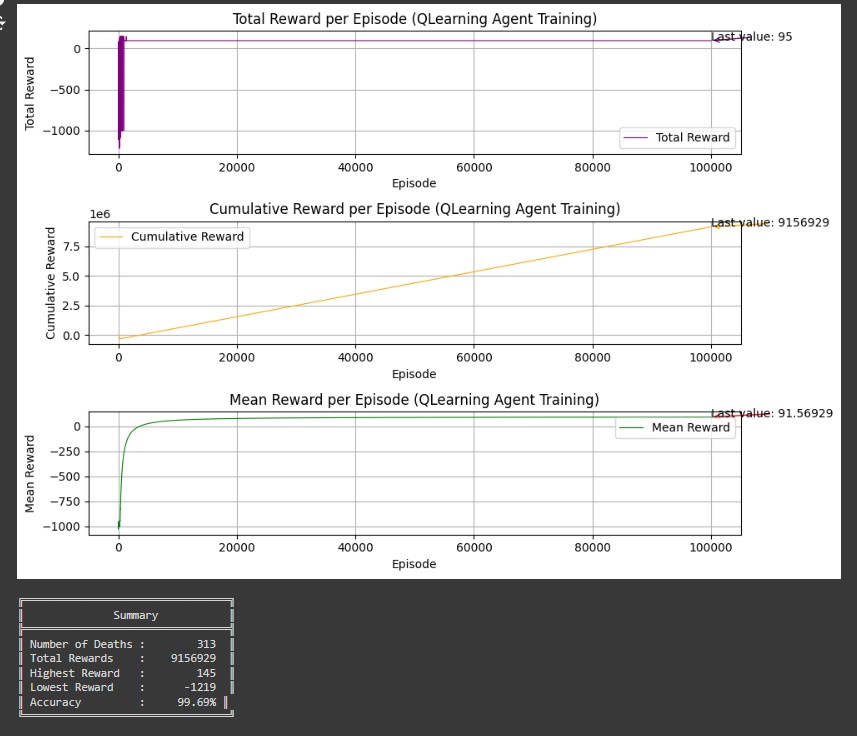
- ذخیره نتایج پاداش‌ها در متغیر total\_rewards.

با ران کردن کد، خروجی ای به صورت شکل صفحه بعد به دست می آید :



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 73 درصد اپیزودها با موفقیت و بدون کشته شدن به پایان رسیده‌اند. با این حال، معیار دقت به تنهایی نمی‌تواند عملکرد مدل را به خوبی نشان دهد، زیرا پس از مدتی، عامل یاد می‌گیرد و بهتر است عملکرد از آن نقطه به بعد ارزیابی شود. نکته مهم این است که روند صعودی نمودار میانگین امتیازها ( نمودار سوم ) همچنان ادامه دارد و هنوز به همگرایی نرسیده است، بنابراین نمایش ریوارد منفی به عنوان آخرین مقدار چندان معتبر نیست. بنابراین قصد داریم الگوریتم را برای 100000 اپیزود نیز بررسی کنیم.

نتایج به صورت صفحه بعد است :



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

همچنین می بینیم که نمودار سوم به مقداری نزدیک 91 همگرا شده است که بهترین سیاستی است که Agent یادگرفته 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:**

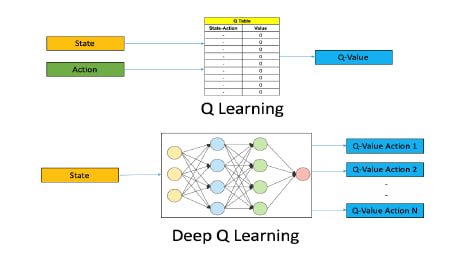
* عملکرد بهتر در محیط‌های با فضای حالت وسیع
* توانایی یادگیری توابع Q-value پیچیده و غیرخطی
* استفاده از تکنیک تجربه تکراری برای کاهش وابستگی به ترتیب تجربیات و بهبود کارایی یادگیری

**معایب:**

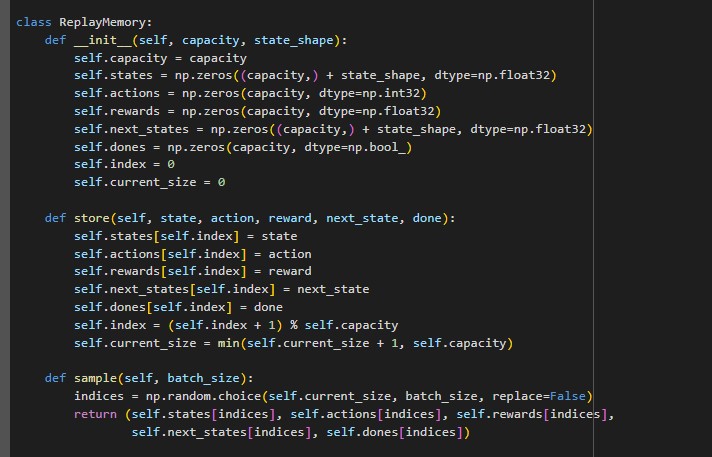
* نیاز به قدرت محاسباتی بالا
* پیچیدگی در تنظیم هایپرپارامترها
* عدم تضمین همگرایی مطمئن

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.

شکل زیر مقایسه ای بین الگوریتم DQN و الگوریتم Q learning است.



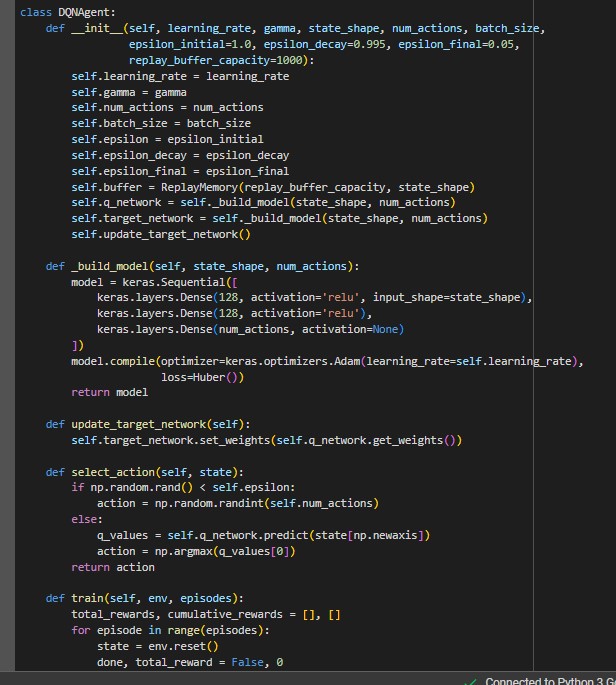
تعاریف اولیه مانند مجموعه‌های Reward Set و Action Set، ایجاد Environment، و قابلیت‌های Agent همانند قبل تعریف شده‌اند. در این بخش، تنها به بررسی بخش‌های جدید اضافه شده در این الگوریتم پرداخته می‌شود (کلاس ReplayMemory)

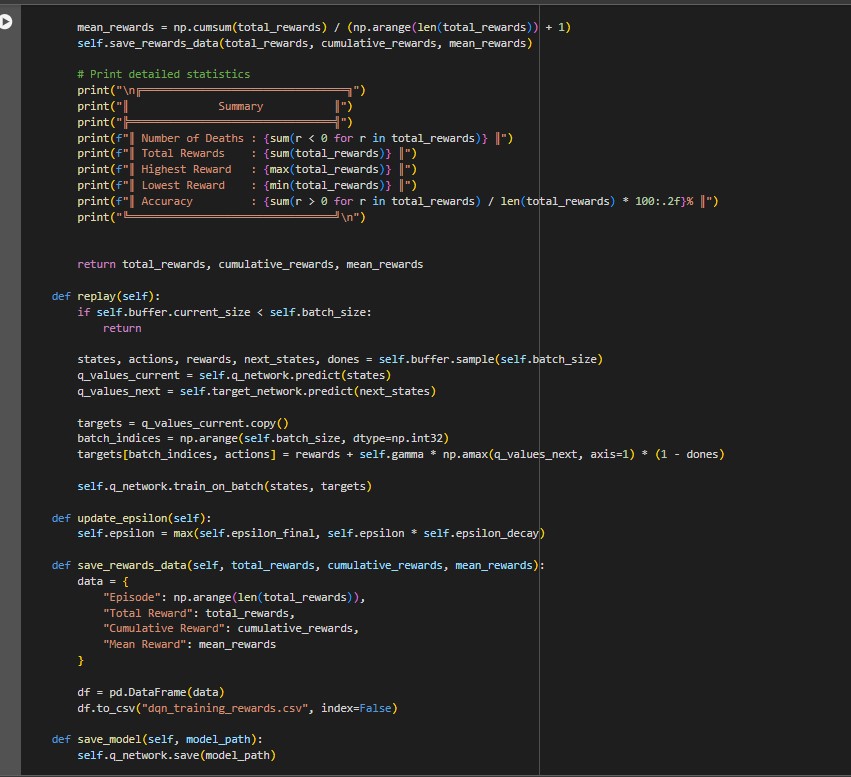


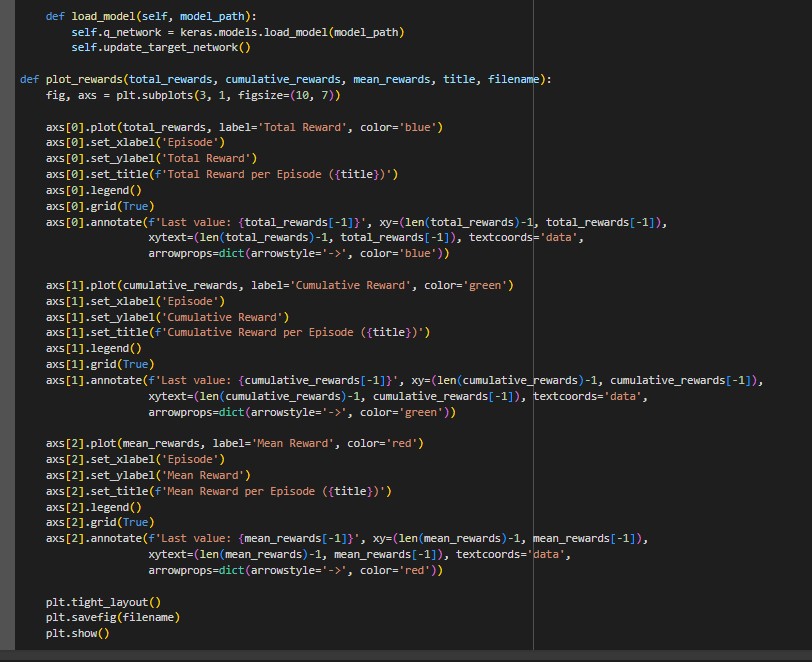
کلاس ReplayMemory برای *ذخیره و مدیریت تجربیات عامل در الگوریتم‌های یادگیری تقویتی مانند DQN طراحی شده است.* این کلاس با ذخیره تجربیات و انتخاب تصادفی آن‌ها برای یادگیری، پایداری و کارایی یادگیری سیاست‌های پیچیده را بهبود می‌بخشد. سه تابع این کلاس (متد) را به اختصار توضیح میدهم.

* **\_\_init\_\_ سازنده:** ظرفیت حافظه و آرایه‌هایی برای ذخیره وضعیت‌ها، اعمال، پاداش‌ها، وضعیت‌های بعدی و شاخص پایان اپیزود را تعریف می‌کن.
* **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.

در ادامه از کد های زیر استفاده میکنیم که در واقع اومذیم کلاس DQNAgent را تعریف کردیم.







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. متد ها و اتربیوت هایی داره که در زیر به اختصار بیان شدند و در شکل های فوق هم قابل مشاهده اند.

\_\_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 . در اخر هم 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.

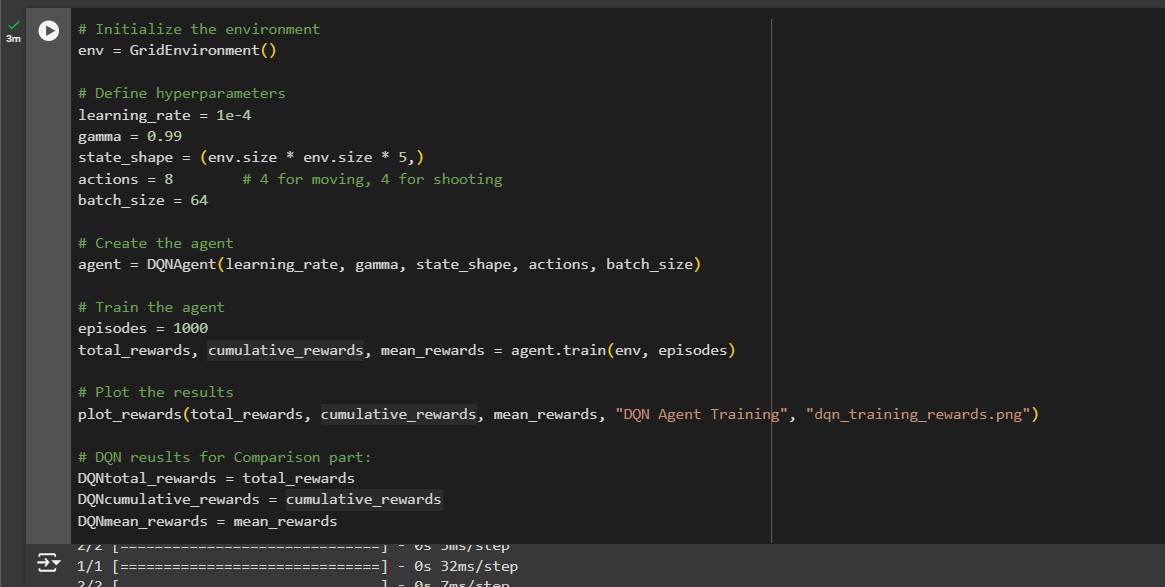
متد save\_rewards\_data: ذخیره داده‌های پاداش آموزشی در فایل CSV.

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.)

حال مقدار دهی های اولیه را انجام میدهیم:

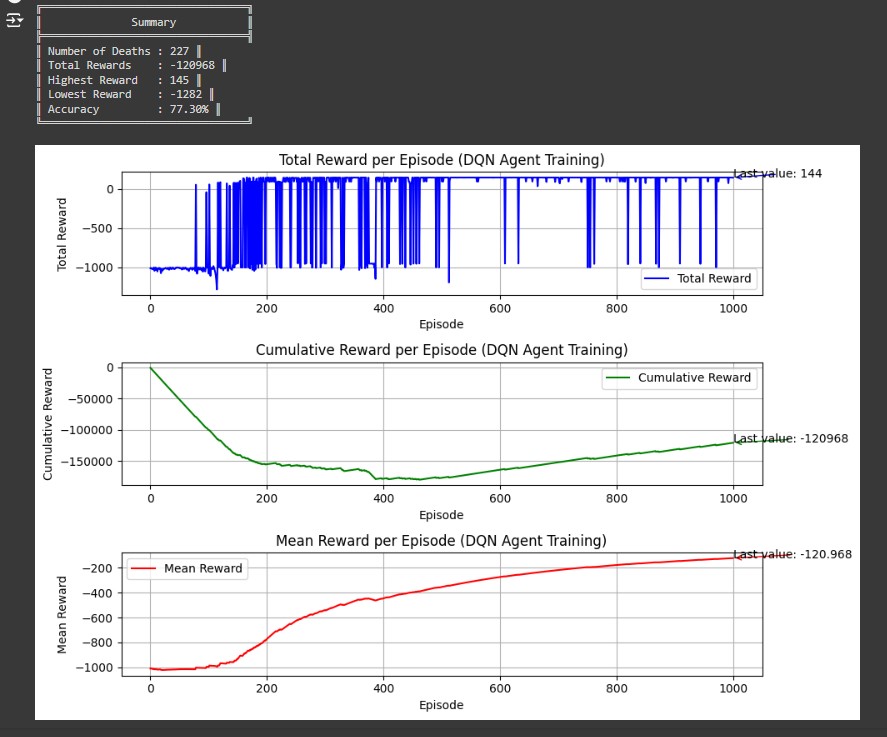


کد فوق با استفاده از یک محیط به نام GridEnvironment شروع می‌شود که به عنوان محیط آموزشی برای الگوریتم DQN عمل می‌کند. سپس، هایپرپارامترهای مختلفی اعمال می‌شوند که شامل نرخ یادگیری (learning\_rate)، ضریب کاهش پاداش (gamma)، شکل حالت (state\_shape)، تعداد اقدامات (actions) و اندازه بچ (batch\_size) می‌باشند.

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.

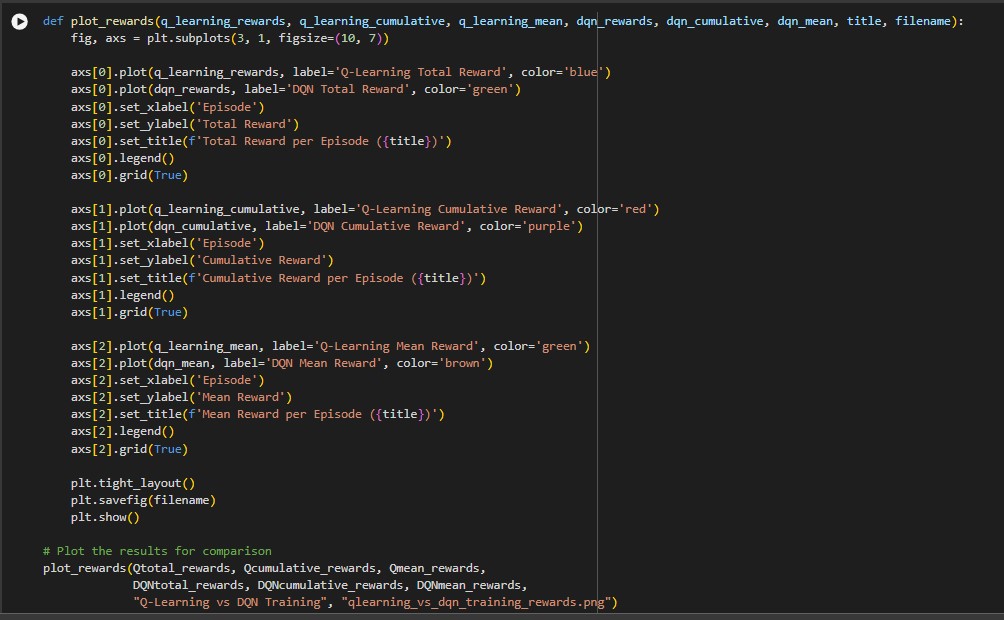
علاوه بر این، مشخص شده است که عامل به طور قطع کشته نشده است و در بسیاری از موارد، در محیط به شدت فعال بوده است. اما، به دلیل حرکات مکرر و دریافت پاداش‌های منفی، نمی‌توان به وضوح تعیین کرد که آیا واقعاً کشته شده یا خیر. ایده‌آل است که نتایج نهایی هر اپیزود را به‌طور جداگانه بررسی کنیم تا بتوانیم به نتیجه‌ای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

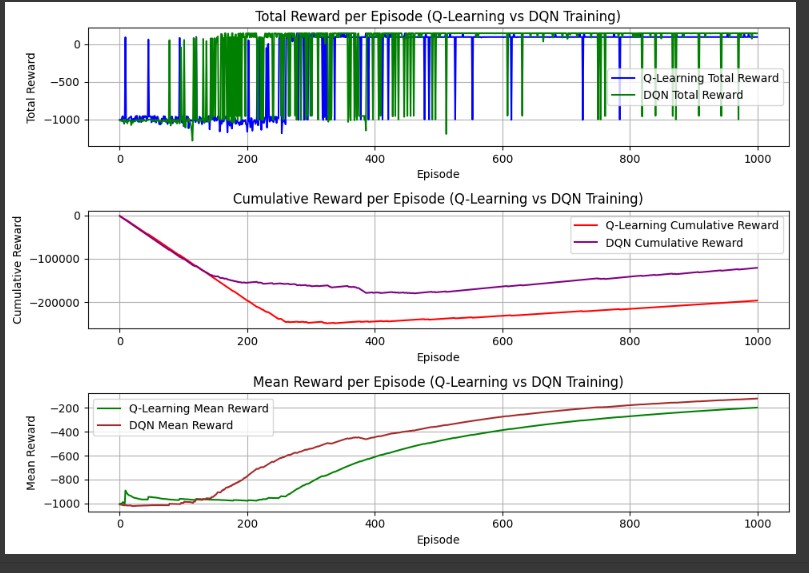
به دلیل زمان طولانی اجرای الگوریتم و محدودیت‌های زمانی، به این نتایج بسنده می‌کنیم. با این حال، الگوریتم DQN همچنان دارای پتانسیل بهبود است، به‌ویژه با کاهش نرخ کاوش و تنظیم نرخ یادگیری برای جلوگیری از گیر افتادن در مینیمم‌های محلی و تعیین تعداد حرکات معقول در هر اپیزود.

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

دو مدل را با یکدیگر مقایسه می کنیم و از کد زیر استفاده می کنیم.





نمودارهای ارائه شده اطلاعات مهمی درباره عملکرد الگوریتم‌های Q-Learning و DQN فراهم می‌کنند. سه نمودار بدست امد و به تحلیل آن ها می پردازیم :

1. **پاداش کل در هر اپیزود**:
   * Q-Learning: در ابتدا با جوایز منفی و نوسانات زیاد مواجه است تا حدود ۳۵۰ اپیزود. بعد از آن، پاداش‌ها به تدریج مثبت می‌شوند و نوسانات کاهش می‌یابد.
   * DQN: نوسانات کمتری در اوایل آموزش دارد و از حدود اپیزود ۱۷۰ پاداش‌های مثبت و پایدارتر می‌شود. این نشان‌دهنده یادگیری سریع‌تر سیاست‌های بهینه است.
2. **پاداش تجمعی**:
   * Q-Learning: در مراحل اولیه به سرعت کاهش می‌یابد و تا حدود اپیزود ۳۵۰ به تدریج افزایش می‌یابد اما همچنان از DQN پایین‌تر است.
   * DQN: در اوایل آموزش کاهش کمتری دارد و از اپیزود ۱۵۰ به بعد به سرعت افزایش می‌یابد و مثبت می‌شود. این الگوریتم در مراحل اولیه عملکرد بهتری دار..
3. ***میانگین پاداش در هر اپیزود****:*
   * Q-Learning: در ابتدا میانگین پاداش بسیار پایین و منفی است و به تدریج از اپیزود ۳۰۰ به بعد افزایش می‌یابد، اما هنوز از DQN پایین‌تر است.
   * DQN: میانگین پاداش در مراحل اولیه پایین است اما از حدود اپیزود ۱۰۰ به بعد به طور مداوم مثبت می‌شود و به ۱۴۵ نزدیک می‌شود، در حالی که Q-Learning به ۹۵ همگرا می‌شود.

**در مجموع، DQN نسبت به Q-Learning با سرعت بیشتری به سیاست‌های بهینه دست می‌یابد، پاداش‌های منفی کمتری دارد و عملکرد بهتری در مراحل اولیه آموزش نشان می‌دهد. DQN با استفاده از شبکه‌های عصبی عمیق و حافظه بازپخش، سیاست‌های بهینه را سریع‌تر و پایدارتر یاد می‌گیرد و برای مسائل پیچیده مناسب‌تر است.**

## بخش ج

تاثیر این نرخ را در هر دو الگوریتم با دقت بررسی می کنیم.

**تاثیر نرخ اکتشاف ϵ بر فرآیند یادگیری مدل Q-Learning**

**نرخ اکتشاف ϵ**:

* یکی از پارامترهای کلیدی در Q-Learning است.
* در ابتدای یادگیری مقدار ϵ برابر با 1.0 است و با هر اپیزود کاهش می‌یابد (ضریب کاهش 0.995).
* ابتدا عامل بیشتر به کاوش محیط می‌پردازد و سپس به بهره‌برداری از سیاست‌های آموخته‌شده.

**تاثیر نرخ اکتشاف بالا**:

* تنوع تجربیات و جلوگیری از گیر افتادن در بهینه‌های محلی.
* نوسانات زیاد و پاداش‌های منفی.
* نیاز به زمان بیشتر برای همگرایی به سیاست بهینه.

**تاثیر نرخ اکتشاف پایین**:

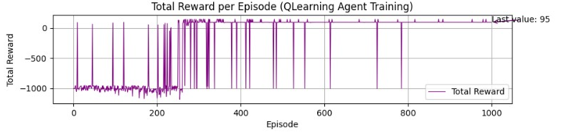
* افزایش بهره‌برداری از سیاست‌های آموخته‌شده.
* پاداش‌های پایدارتر و مثبت‌تر.
* امکان گیر افتادن در نقاط بهینه محلی.

**تنظیم بهینه ϵ**:

* تعادل بین کاوش و بهره‌برداری.
* تاثیر چشمگیر بر کیفیت یادگیری و عملکرد نهایی عامل.
* در نمودار، در مراحل اولیه با نرخ ϵ بالا، پاداش‌ها نوسانات زیادی دارند و معمولاً منفی هستند. با کاهش نرخ ϵ، پاداش‌ها پایدارتر و مثبت‌تر می‌شوند.

پس به طور خلاصه نرخ اکتشاف ϵ در الگوریتم Q-Learning یکی از پارامترهای کلیدی است که تعادل بین کاوش محیط و بهره‌برداری از سیاست‌های آموخته‌شده را تعیین می‌کند. در ابتدا، ϵ برابر با 1.0 است و به تدریج با ضریب 0.995 کاهش می‌یابد، تا در پایان 1000 اپیزود به مقدار کمی برسد. نرخ اکتشاف بالا در مراحل اولیه یادگیری به عامل کمک می‌کند تا تنوع تجربیات بیشتری کسب کند و از گیر افتادن در نقاط بهینه محلی جلوگیری کند، اما ممکن است منجر به ناپایداری و کاهش پاداش‌های کسب‌شده شود. در مقابل، نرخ اکتشاف پایین در مراحل پایانی بهره‌برداری بیشتر از سیاست‌های آموخته‌شده را تشویق می‌کند و به همگرایی سریع‌تر به سیاست بهینه کمک می‌کند، اما ممکن است فرصت‌های کاوش محیط و کشف سیاست‌های بهتر را محدود کند. بنابراین، تنظیم بهینه نرخ اکتشاف ϵ برای تعادل مناسب بین کاوش و بهره‌برداری بسیار مهم است تا بهترین عملکرد در طول فرآیند یادگیری حاصل شود.

امتیاز های مدل برای Q learning به صورت زیر است ( در بخش آ هم بیان شد.)



مشاهده می کنیم که در مراحل ابتدایی یادگیری با نرخ ϵ بالا، پاداش‌ها نوسان زیادی دارند و معمولاً منفی هستند زیرا عامل بیشتر به کاوش و اعمال تصادفی می‌پردازد. با کاهش نرخ ϵ، عامل به تدریج به بهره‌برداری از سیاست‌های آموخته‌شده می‌پردازد و پاداش‌ها پایدارتر و مثبت‌تر می‌شوند. در حدود اپیزود 550، پاداش‌ها به حالت پایدار و مثبت می‌رسند که نشان‌دهنده همگرایی به سیاست بهینه است. انتخاب و تنظیم صحیح نرخ ϵ در الگوریتم Q-Learning بسیار مهم است؛ نرخ بالای اکتشاف در ابتدا برای درک بهتر محیط و جمع‌آوری تجربیات متنوع و کاهش تدریجی آن برای بهره‌برداری بیشتر از سیاست‌های یادگرفته‌شده و بهبود عملکرد یادگیری است.

**تاثیر نرخ اکتشاف ϵ بر فرآیند یادگیری مدل DQN**

**مراحل ابتدایی**:

* نرخ اکتشاف بالا منجر به نوسانات زیاد و پاداش‌های منفی.
* کمک به شناخت بهتر محیط و کسب تجربیات متنوع‌تر.

**میانه فرآیند یادگیری**:

* کاهش نرخ اکتشاف و افزایش بهره‌برداری از سیاست‌های آموخته‌شده.
* کاهش نوسانات پاداش و افزایش پاداش‌های مثبت.

**مراحل پایانی**:

* نرخ اکتشاف پایین، پاداش‌های پایدارتر و مثبت‌تر.
* بهره‌برداری بیشتر از سیاست‌های آموخته‌شده.

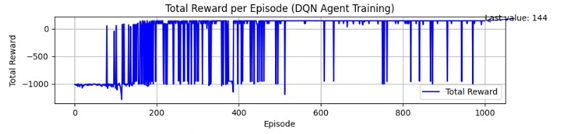
**تنظیم مناسب نرخ اکتشاف**:

* نقش مهم در بهبود فرآیند یادگیری و عملکرد نهایی عامل.
* کاهش تدریجی نرخ اکتشاف از 1 به 0.05 طی 600 اپیزود.

**نتیجه‌گیری**

* نرخ اکتشاف ϵ بالا در مراحل اولیه به عامل کمک می‌کند تا محیط را بهتر بشناسد و تجربیات متنوع‌تری کسب کند.
* کاهش تدریجی نرخ اکتشاف به بهره‌برداری بهتر از سیاست‌های آموخته‌شده و همگرایی به سیاست بهینه کمک می‌کند.
* تنظیم صحیح نرخ اکتشاف باعث بهبود عملکرد و پایداری فرآیند یادگیری می‌شود.

شکل زیر امتیاز دهی مدل DQN را در هر اپیزود نشان می دهد:



در اوایل فرآیند یادگیری در الگوریتم DQN، نرخ اکتشاف بالا است و پاداش‌ها نوسانی و عمدتاً منفی هستند. این به دلیل انجام اعمال تصادفی توسط عامل برای کاوش محیط و کسب تجربیات متنوع‌تر است. با گذشت زمان و کاهش نرخ اکتشاف، عامل شروع به بهره‌برداری از سیاست‌های آموخته شده می‌کند، که منجر به کاهش نوسانات پاداش و افزایش پاداش‌های مثبت می‌شود. این تغییر نشان‌دهنده بهبود عملکرد عامل است، هرچند که هنوز ممکن است پاداش‌های منفی نیز دیده شود و همگرایی کامل رخ نداده باشد.

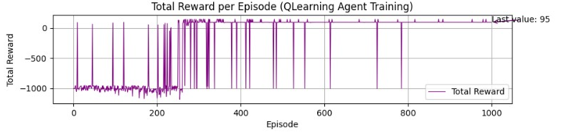
در اواخر فرآیند یادگیری، نرخ اکتشاف به پایین‌ترین مقدار خود می‌رسد و پاداش‌ها پایدارتر و مثبت‌تر می‌شوند. عامل بیشتر به بهره‌برداری از سیاست‌های آموخته شده می‌پردازد، اما همچنان ممکن است به سیاست بهینه همگرا نشده باشد. با این حال، نمودار نشان می‌دهد که عملکرد عامل به تدریج بهبود می‌یابد و پاداش‌های بالاتری کسب می‌کند. به طور کلی، تنظیم مناسب و کاهش تدریجی نرخ اکتشاف نقش مهمی در بهبود فرآیند یادگیری و عملکرد نهایی عامل دارد. پس از حدود 600 اپیزود، مقدار اپسیلون به 0.05 همگرا می‌شود و عامل از حالت اکتشاف به بهره‌برداری منتقل می‌شود. تنظیم بهتر مدل و تعریف محدودیت‌هایی برای تعداد گام‌های مجاز می‌تواند به بهبود عملکرد و همگرایی کمک کند.

## بخش د

**تحلیل تعداد اپیزودها برای دستیابی به عملکرد پایدار**

**Q-Learning**:

* نیاز به حدود 300 اپیزود برای دستیابی به عملکرد پایدار و مثبت.
* پس از 520 اپیزود به طور کامل بازی را فراگرفته و پاداش منفی دریافت نمی‌کند.
* فرآیند یادگیری طولانی و ناپایدار است و ممکن است به سیاست بهینه نرسد.



**DQN**:

* تنها به حدود 100 اپیزود برای رسیدن به عملکرد پایدار و مثبت نیاز دارد.
* سریع‌تر به سیاست‌های بهینه دست می‌یابد و عملکرد پایدارتر و بهتری دارد.
* نمی‌توان دقیقاً گفت که پاداش‌های منفی به دلیل کشته شدن عامل یا حرکات بیش از حد ایجاد شده‌اند.

**تحلیل کارایی مدل‌ها در دستیابی به سیاست بهینه**

**Q-Learning**:

* نوسانات و پاداش‌های منفی در مراحل اولیه
* به تدریج به سیاست‌های بهینه نزدیک می‌شود، اما فرآیند طولانی و ناپایدار است
* ممکن است به بهترین سیاست دست نیابد

**DQN**:

* بهره‌گیری از شبکه‌های عصبی عمیق و حافظه بازپخش
* سیاست‌های بهینه را سریع‌تر و با ثبات بیشتری یاد می‌گیرد.
* نمودارهای پاداش تجمعی و میانگین پاداش نشان‌دهنده عملکرد برتر DQN است.

***نتیجه‌گیری:*** دو نتیجه زیر را می توان گرفت

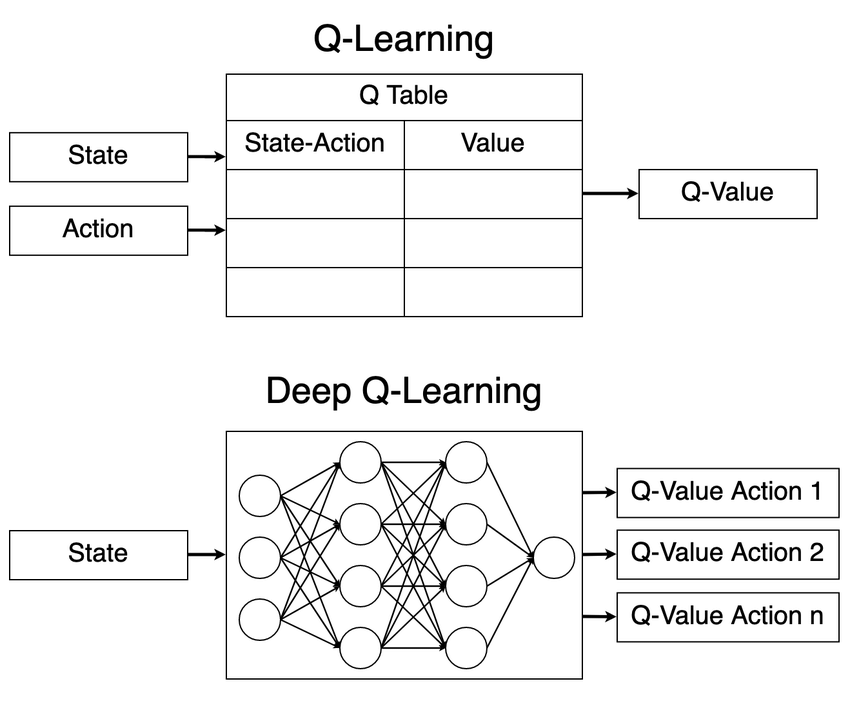
* **DQN**: سریع‌تر، پایدارتر و مؤثرتر از Q-Learning عمل می‌کند و گزینه بهتری برای مسائل پیچیده با فضای حالت بزرگ است.
* **Q-Learning**: فرآیند یادگیری طولانی‌تر و ناپایدارتر، با نوسانات بیشتر در مراحل اولیه.

DQN با استفاده از تکنیک‌های پیشرفته در یادگیری تقویتی، به ویژه برای مسائل پیچیده، عملکرد بهتری نسبت به Q-Learning دارد.

## بخش ه

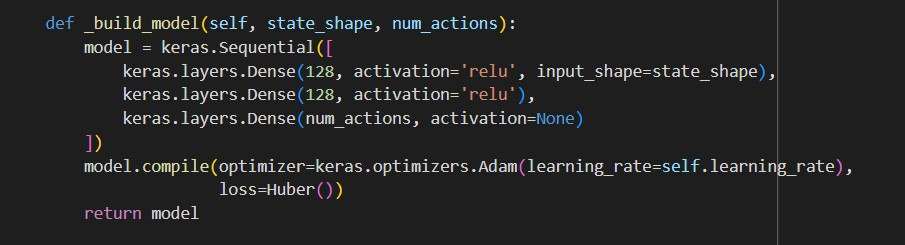
در الگوریتم Deep Q-Networks (DQN)، شبکه عصبی به گونه‌ای طراحی شده است که توابع Q-value را برای محیط‌های پیچیده با فضای حالت بزرگ تخمین بزند. این شبکه شامل لایه‌های مخفی و توابع فعال‌سازی است.

ساختار کلی این شبکه به صورت زیر است :



**ساختار مدل**:

کد پایتون برای پیاده سازی آن به صورت زیر است :



از Keras برای ساخت مدل استفاده می‌شود.

* **تابع \_build\_model**: دو ورودی state\_shape (شکل ورودی حالت‌ها) و num\_actions (تعداد اکشن‌ها) دارد.
* **لایه‌ها**:
  + **لایه اول**: Dense با 128 نورون و تابع فعال‌سازی ReLU
  + **لایه دوم**: Dense با 128 نورون و تابع فعال‌سازی ReLU
  + **لایه سوم**: Dense با تعداد نورون برابر با num\_actions و خروجی خطی.

**کامپایل مدل**: از کد هم قابل مشاهده است که :

* **بهینه‌ساز**: Adam با نرخ یادگیری تنظیم شده
* **تابع هزینه**: هابر (Huber loss) برای کاهش تاثیر نویزهای بزرگ

**دلایل انتخاب معماری**:

1. **توابع فعال‌سازی ReLU**:
   * محاسبات سریع‌تر نسبت به سیگموئید و تانژانت هایپربولیک
   * عدم مشکل اشباع و مقیاس‌پذیری بهتر
2. **تعداد نورون‌ها**:
   * 128 نورون در هر لایه مخفی تعادل مناسبی بین پیچیدگی و قابلیت‌های محاسباتی ایجاد می‌کند
   * جلوگیری از بیش‌برازش و یادگیری ویژگی‌های پیچیده
3. **تابع هزینه هابر**:
   * ترکیب خطای مطلق و خطای مربعی
   * کاهش حساسیت به نقاط پرت و کمک به پایداری یادگیری

**مزایا**:

* **یادگیری ویژگی‌های پیچیده**: شبکه قادر به یادگیری و تقریب توابع Q-value پیچیده است
* **لایه‌های مخفی متعدد**: توانایی یادگیری روابط پیچیده بین وضعیت‌ها و اعمال
* **پایداری و کارایی**: استفاده از تابع Huber به بهبود پایداری و کاهش نوسانات در به‌روزرسانی وزن‌ها کمک می‌کند.

**نرخ کاوش (Exploration)**:

* کاهش تدریجی نرخ کاوش که به دستیابی به سیاست بهینه با امتیاز 145 کمک کرده است.

این معماری به دلیل سادگی و کارایی بالا، یکی از معماری‌های رایج در مسائل یادگیری تقویتی پیچیده مانند بازی‌های ویدیویی و شبیه‌سازی‌های پیچیده است.