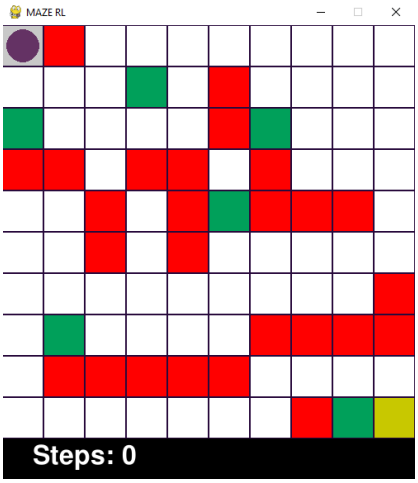
برنامه نوشته شده در قالب jupyter notebook ارائه شده است که می توان بعد از اجرای تعاریف کلاس ها و آموزش الگوریتم ، با دستور rl.play(True, True) ، مسیر نهایی را صرف نظر از قابلیت epsilon (جستجوگری کمتر) در محیط pygame مشاهده کرد.



در حل مسئله خود Action ها را اعمالی در نظر گرفتیم که agent می تواند انجام دهد

{Up , Right , Down , Left}

(البته حرکاتی که باعث شود agent از محیط خارج شود یا در block ها قرار گیرد را در مراحل در نظر نمی گیریم.)

Reward را نیز به صورت زیر در نظر گرفتیم :

- در صورتی که agent به خانه ای که قبلا ان را مشاهده کرده برگردد (-10)
- در صورتی که agent ، یک پرچم را دریافت کند (50)
- در صورتی که agent به خانه target برسد ولی همه پرچم ها را دریافت نکرده باشد (-400)
- در صورتی که agent همه پرچم ها را دریافت کرده باشد و به target برسد (100)
- و اگر شرایط فوق برقرار نبود (برای مجبور کردن agent به کم کردن steps) (-1)

State را نیز به دو صورت تعریف کردیم (برای بعضی مقادیر هر دو state بررسی شده اند.):

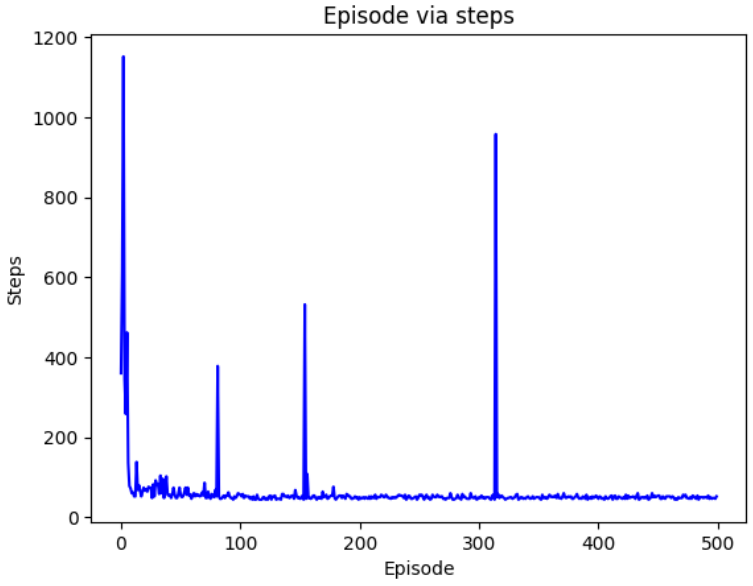
۱- (Flags و موقت مکانی agent) : در واقع Flags در اینجا tuple مقادیر Boolean می باشد و با ترتیب خاصی که در ابتدا مشخص می شود ، هر عضو آن نشان دهنده این است که ایا پرچم متناظر برداشته شده یا خیر . ((0, 2, (False, True, False, False, False, False))

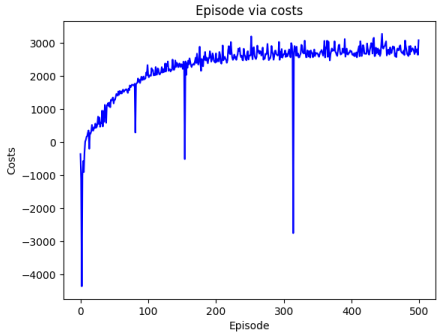
۲- (Flags_count و موقت مکانی agent) : در این روش Flags_count صرفا تعداد پرچم ها را نشان می دهد و طبق بررسی های انجام شده تعداد state کم تری نسبت به روش قبلی دارد و جواب تقریبا مشابه ای را دارد. (0, 2, 1)

روش دیگر هم برای کاهش state ها می توانستیم انجام دهیم به این صورت که خانه های جدول را به مناطقی ۲ * ۲ یا ۳ *۳ یا ... تبدیل می کردیم و دیگر موقعیت مکانی agent در واقع منطقه مکانی agent گفته می شد . روشن است که این روش state بسیار کمتری را نسبت به قبلی ها دارد و احتمالا طبق جدول داده شده می توانست نتیجه مشابه ای را داشته باشد اما از انجا که روش ۲ به اندازه کافی خوب بود ، نیازی به استفاده از این روش احساس نشد.

درنتیجه روشن است که Goal State ، رسیدن agent به target بدون flags باقی مانده می باشد.

قسمت اول : در این قسمت با روش ۱ state و gamma = .9 و alpha = .1 ، rl آموزش دیده شد و نتایج زیر به دست امد :





که rl.play نیز نتیجه زیر را داد :

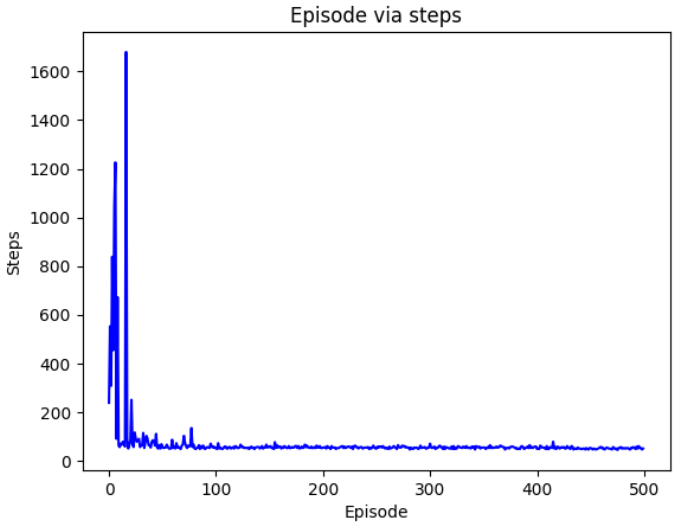
```
(44, 2601.8491667560884)
```

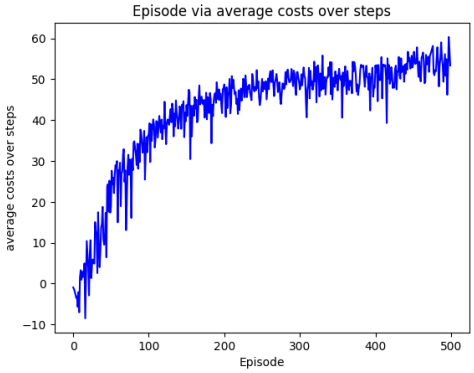
که متغیر اول نشان دهنده تعداد step و متغیر دوم نشان دهنده مجموع cost در طول مسیر می باشد.

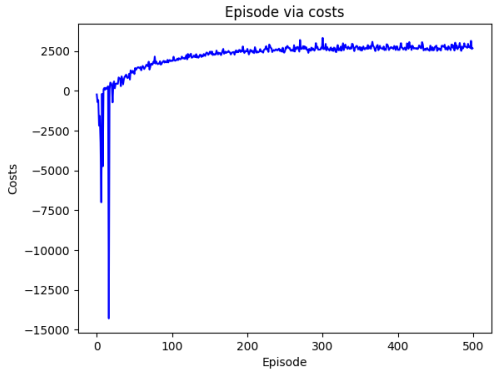
همانطور که مشخص است نمودار ها در بعضی نقاط (تعداد کم) نتیجه بسیار دوری را ثبت کردند و این به دلیل وجود جستجو گری (epsilon) در الگوریتم مطروحه می باشد.

قسمت دوم : (only_count)

اکنون only_count را فعال می کنیم. همانطور که گفته بودیم با این کار تعداد state ها کاهش پیدا می کند و نتیجه ای مشابه قسمت اول را خواهیم دید و اگر epsilon را در نظر نگیریم ، الگوریتم مطروحه کمی زودتر به نتیجه مورد نظر می رسد.







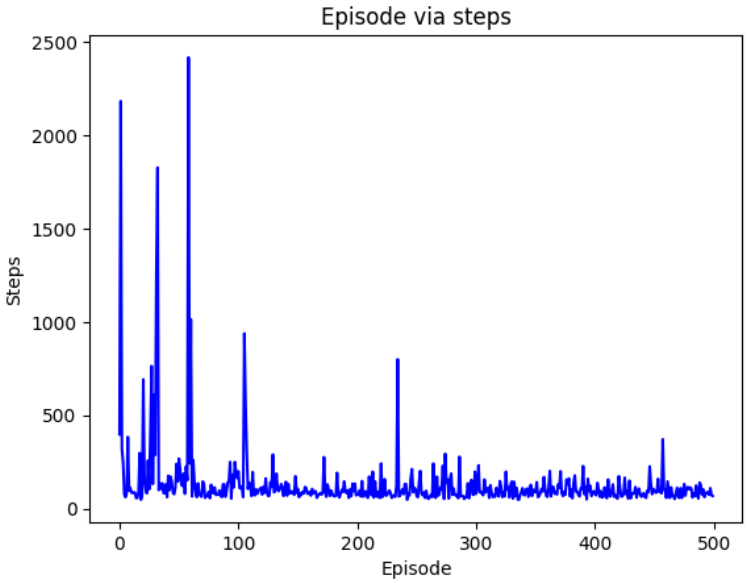
که rl.play نیز نتیجه زیر را داد :

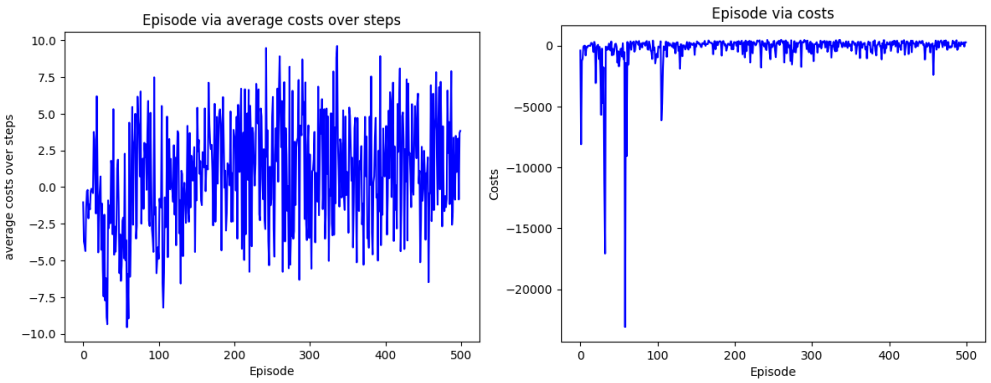
```
(44, 2552.5768610527157)
```

و نکته مهم دیگر این است که در این نمودار ها تعداد نوسان ها به شدت کاهش پیدا کرده است.

قسمت سوم : (gamma = 0.25 alpha = 0.1)

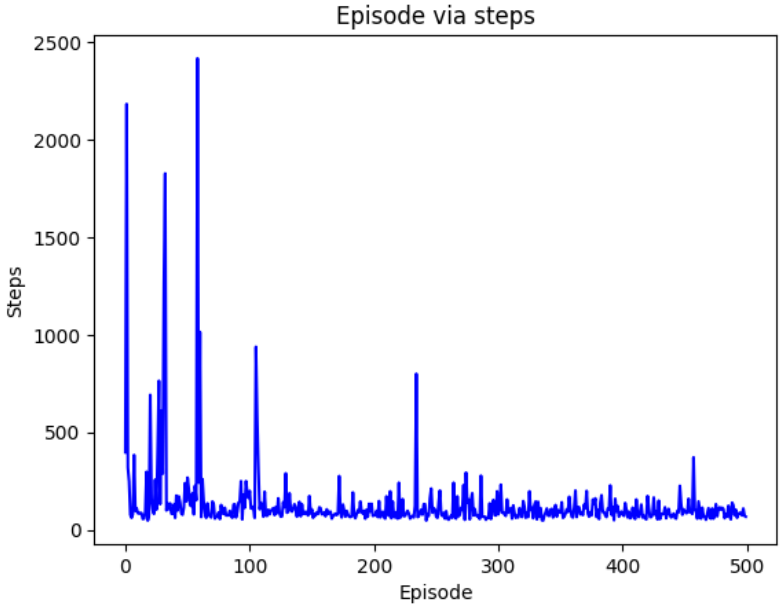
همانطور که مشخص است در این ازمایش متغیر گاما را نسبت به قسمت اول کاهش دادیم و نتایج زیر مشاهده شد:

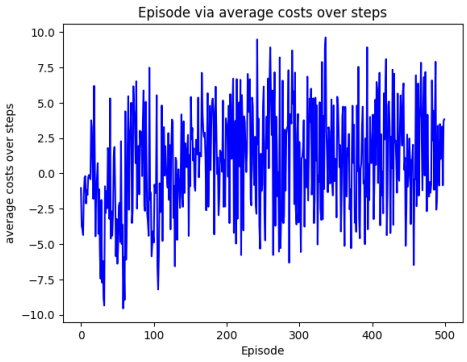


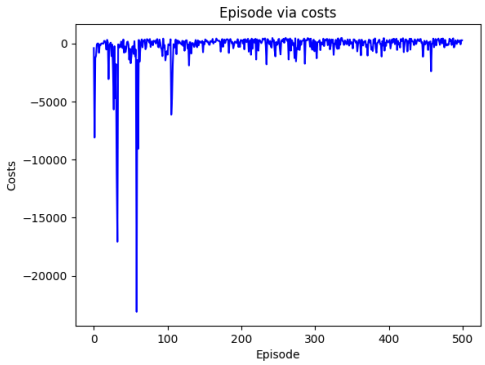


که روشن است که به نتیجه ای معقول نرسیده ایم و نوسان ها به شدت زیاد است به طوری که در rl.play ، agent در یک حلقه گیر می کند چرا که با کم کردن gamma ، انتخابات آینده تاثیر کمی بر روی انتخاب حال دارد همچنین با این کار تعداد state هایی که در q table مقدار دهی می شوند بیشتر خواهد بود چرا که حتی امکان برگشت از مسیر و بررسی state هایی که در حالت قبلی دیده نمی شدند (مثلا flag آخر برداشته می شد و به اول جدول بر می گشت !) وجود داشت. پس به هیچ وجه این gamma مناسب نبود.

همچنین برای اطمینان از این موضوع حتی state 2 نیز مورد بررسی قرار گرفت و نتیجه ای مشابه را دیدیم.



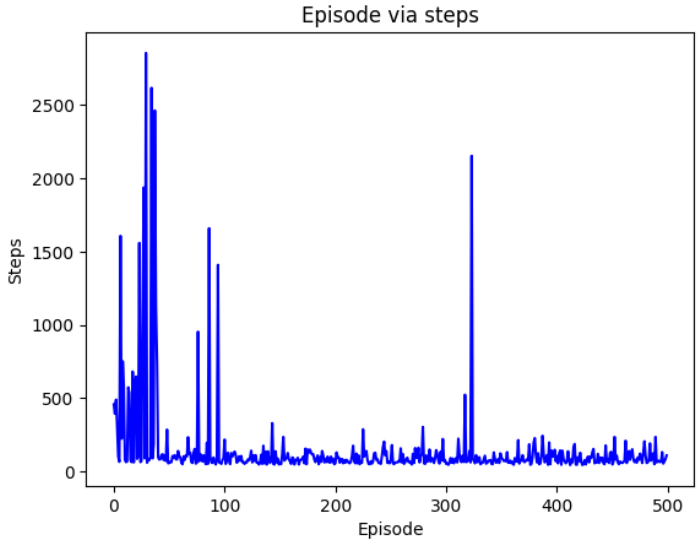


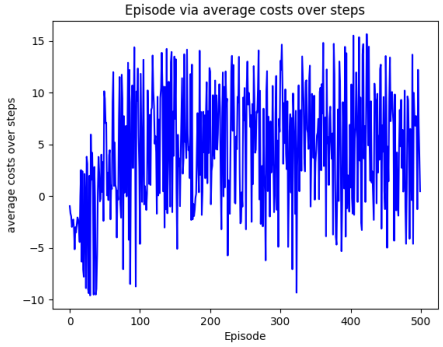


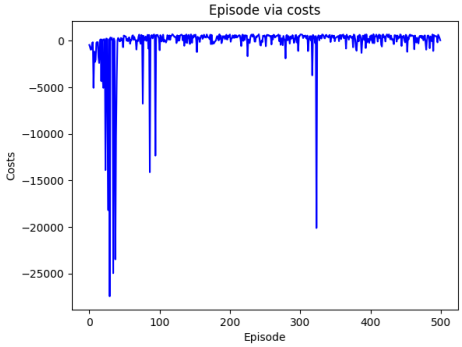
همانطور که مشخص است حتی با این نوع state هم ، تعداد state ها بالا بوده و به همین سبب agent مخصوصا در rl.play ، به خوبی بازی نمی کند.

قسمت چهارم : (gamma = 0.5)

هنوز هم گاما مقدار کمی را دارد و state های زیادی را مشاهده میکنیم و مانند قسمت قبل rl.play ، در یک حلقه گیر می کند و نشان دهنده شباهت q ها در بعضی مراحل است .

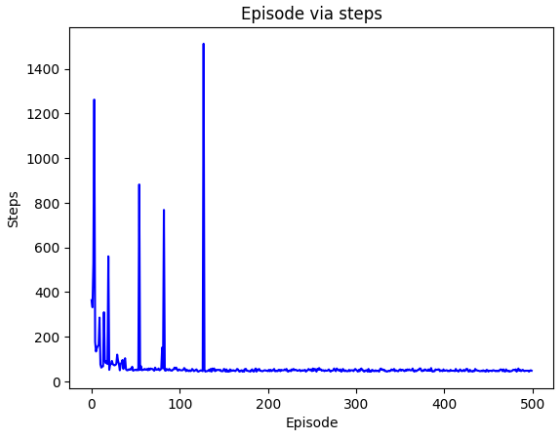


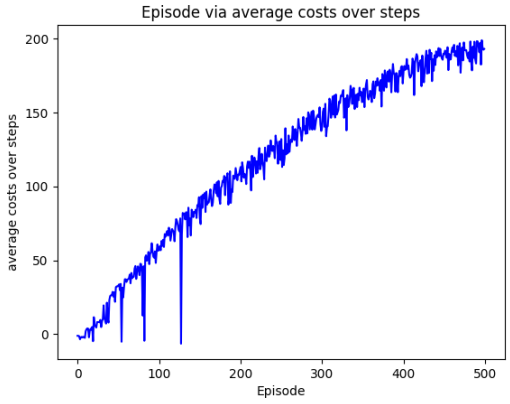


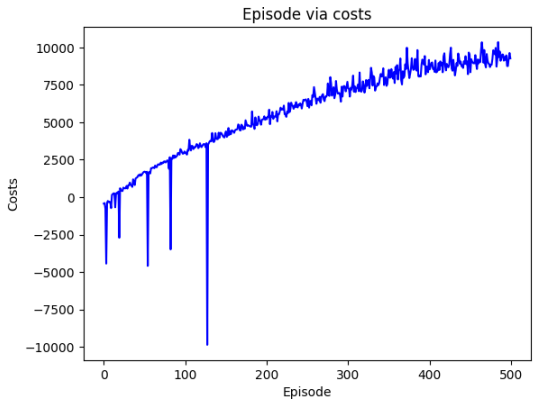


قسمت پنجم : (gamma = 1 alpha = 0.1)

همانطور که مشخص است این گاما مقدار خوبی را دارد و مانند تست های اول و دوم نتیجه خوبی را نشان می دهد البته روشن است که با این گاما تاثیر q ها بر هم بیشتر شده و در نتیجه مجموع cost ها بیشتر از قبل خواهد بود .

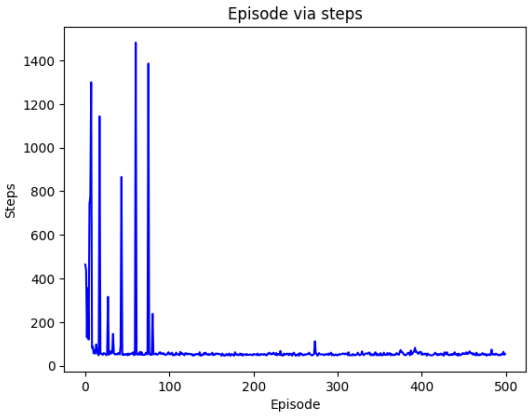


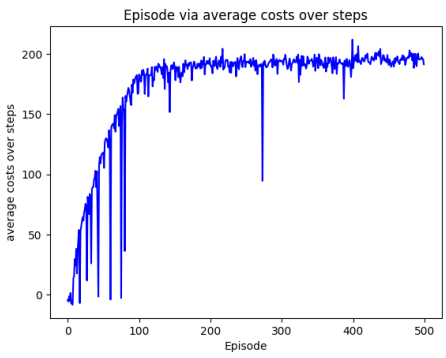


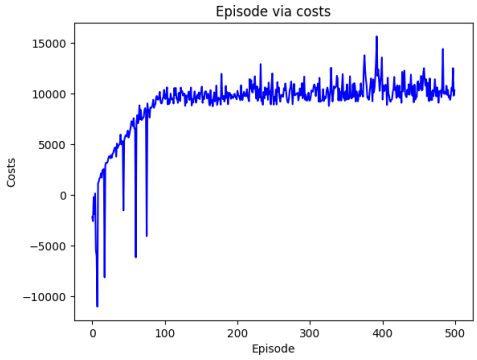


قسمت ششم : (gamma = 1 alpha =0.5)

همانطور که مشخص است با افزایش alpha سرعت یادگیری افزایش می یابد و می توانیم در کم تر از ۱۰ آموزش ، مسیر بهینه را پیدا کنیم برخلاف تست های قبلی که نیاز به حداقل ۵۰ آموزش داشتیم.





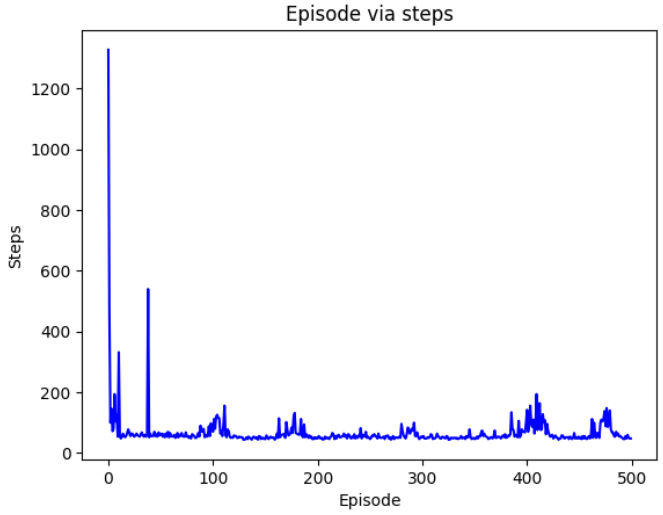


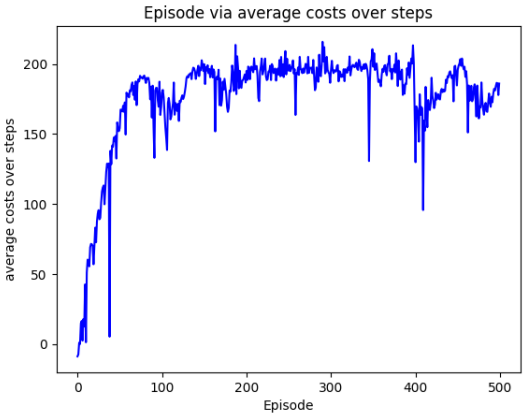
همانطور که از نمودار steps مشخص است نوسان کمی را داریم و این می تواند نشان دهنده جستجوگری کم نیز باشد همانطور که در rl.play با غیر فعال کردن epsilon و کاهش جستجوگری ، برای مدتی در حلقه هایی گیر می کنیم و نتیجه خوبی را دریافت نمی کنیم . در واقع در مسیر به دلیل زیاد بودن alpha، در بهینه های محلی گیر می کند.

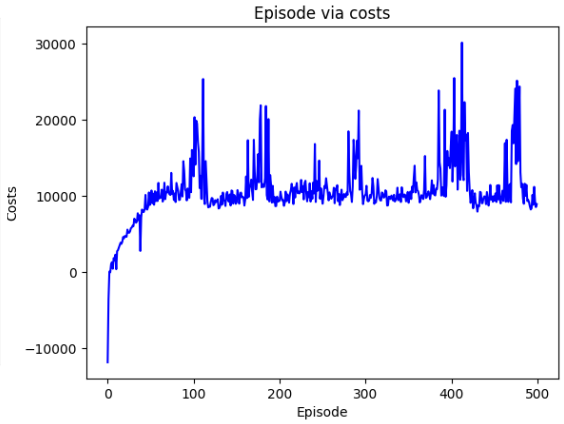
(66, 12652.454342982486)

قسمت هفتم : (gamma = 1 alpha =0.8)

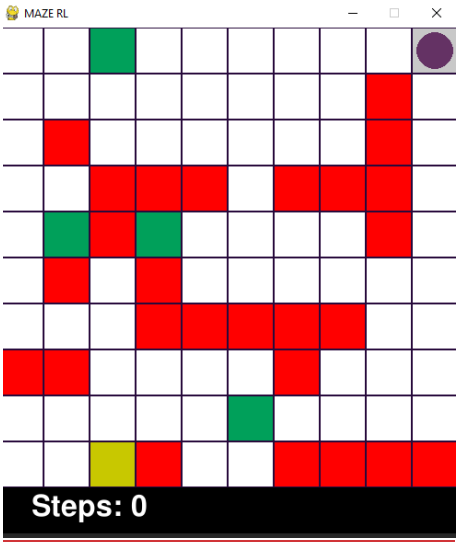
این ازمایش هم مانند قسمت قبل نتیجه خوبی را ارائه می دهد اما اگر epsilon را خاموش کنیم ، جستجوگری کم شده و در rl.play در بیشتر مواقع در بهینه محلی گیر می کند اما نسبت به قسمت قبل کمی نوسان داشته و در بعضی مواقع جواب صحیحی را خروجی می دهد.

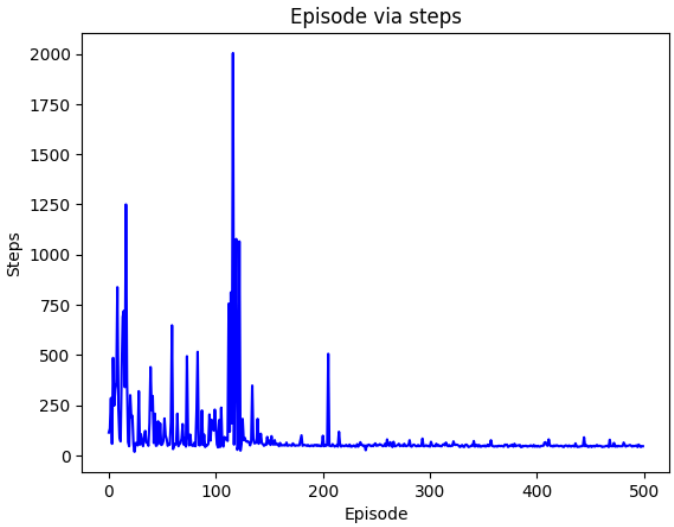


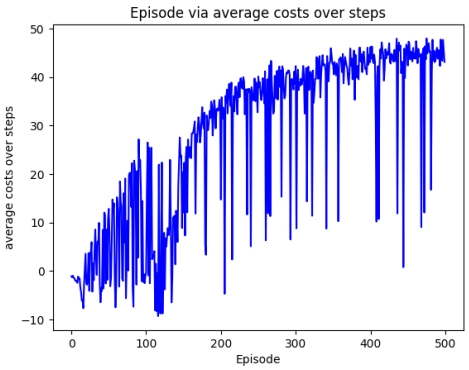


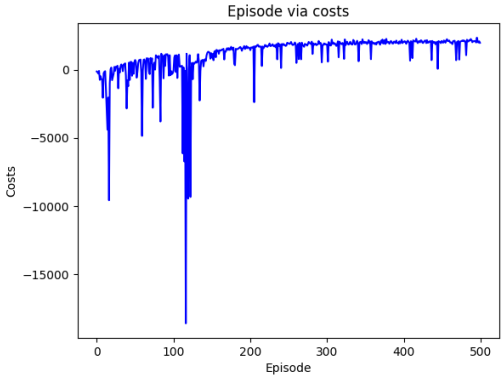


قسمت نهایی : تست با Maze دیگر (alpha = 0.1 gamma = 0.9)









این Maze نسبت به Maze قبلی سخت تر بوده و تعداد State بیشتری را نیاز داشت اما الگوریتم طراحی شده توانست بعد از ۱۲۰ مرحله به حالت بهینه خود برسد به طوری که در rl.play نیز نتیجه زیر را داشته است.

```
(42, 2005.6264508206866)
```

پس همانطور که بیان شد گاما و الفا بیان شده بهتر از دیگر گاما و الفا های تست شده هستند و حتی در جدول دیگر نیز به خوبی کار می کنند.