

دانشکده مهندسی برق

یادگیری تقویتی در کنترل

تمرین اول: شبیه‌سازی مسئله MAB

استاد: دکتر سعید شمقدری

دانشجو: سیده ستاره خسروی

پائیر 1403

چکیده

در این تمرین به پاسخ سوال آخر تمرین سری دوم و بخش کدنویسی تمرین پرداخته می‌شود.

واژه‌های کلیدی: یادگیری تقویتی، راهزن چند دست

# فهرست مطالب

عنوان صفحه

[فهرست مطالب ‌ب](#_Toc179832718)

[فهرست تصاویر و نمودارها ‌ج](#_Toc179832719)

[فصل 1: شبیه‌سازی مسئله راهزن چنددست 1](#_Toc179832720)

[1‌.1‌ مقدمه 1](#_Toc179832721)

[1‌.2‌ پیاده سازی GreedyAgent 1](#_Toc179832722)

[1‌.3‌ پیاده سازی EpsilonGreedyAgent 5](#_Toc179832723)

[1‌.4‌ پیاده سازی EpsilonGreedyAgentConstantStepsize 7](#_Toc179832724)

[1‌.5‌ پیاده سازی UCBAgent 9](#_Toc179832725)

# فهرست تصاویر و نمودارها

عنوان صفحه

[شکل ‏1‑1: فراخوانی کلاس‌ها و کتابخانه‌ها 1](#_Toc179832727)

[شکل ‏1‑2: کد argmax 2](#_Toc179832728)

[شکل ‏1‑3: صحت عملکرد argmax 2](#_Toc179832729)

[شکل ‏1‑4: دایرکتوری کد نویسی 3](#_Toc179832730)

[شکل ‏1‑5: الگوریتم Greedy 3](#_Toc179832731)

[شکل ‏1‑6: کد کلاس GreedyAgent 4](#_Toc179832732)

[شکل ‏1‑7: خروجی GreedyAgent 4](#_Toc179832733)

[شکل ‏1‑8: کد بخش EpsilonGreedyAgent 5](#_Toc179832734)

[شکل ‏1‑9: مقایسه‌ی epsilon های مختلف 6](#_Toc179832735)

[شکل ‏1‑10: مقایسه‌ی دو اجرای متفاوت 7](#_Toc179832736)

[شکل ‏1‑11: کد FixedStepSize 8](#_Toc179832737)

[شکل ‏1‑12: نمودار FixedStepSize 8](#_Toc179832738)

[شکل ‏1‑13: کد UCB 9](#_Toc179832739)

[شکل ‏1‑14: بخش اول کد 10](#_Toc179832740)

[شکل ‏1‑15: بخش آخر کد UCB 11](#_Toc179832741)

[شکل ‏1‑16: بخش دوم کد UCB 11](#_Toc179832742)

[شکل ‏1‑17: خروجی UCB و EpsilonGreedy 12](#_Toc179832743)

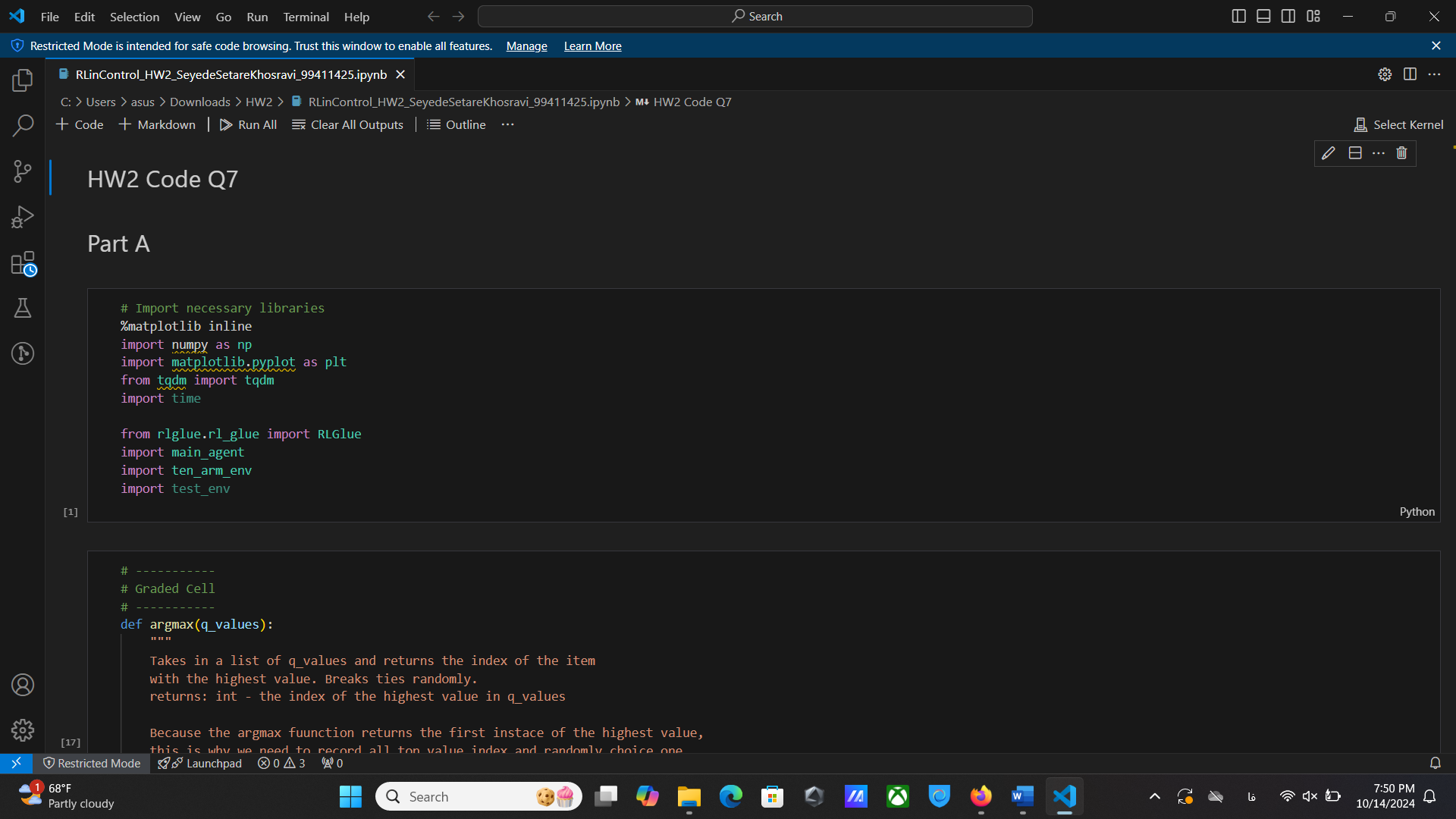
[شکل ‏1‑18: خروجی UCB و EpsilonGreedy پس از 2000 اجرا 13](#_Toc179832744)

# شبیه‌سازی مسئله راهزن چنددست

## مقدمه

سوال هفتم تمرین مرتبط با شبیه‌سازی مسئله راهزن چند دست برای 10 بازو است. در ادامه به حل آن می‌پردازیم.

## پیاده سازی GreedyAgent

ابتدا مطابق کد زیر، کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم. کلاس‌هایی نیز لازم است که فراخوانی شوند، کد این کلاس‌ها را از گیتهاب پیدا کرده و در همان مسیری که می‌خواهیم کدنویسی کنیم، قرار می‌دهیم تا فراخوانی آن2ها با مشکل مواجه نشود.

شکل ‏1‑1: فراخوانی کلاس‌ها و کتابخانه‌ها

سپس کد argmax مطابق با صورت تمرین نوشته شده و آن را تست می‌کنیم که پاسخ صحیح برگرداند.

A screenshot of a computer

Description automatically generated

شکل ‏1‑2: کد argmax

A screenshot of a computer

Description automatically generatedدر ادامه با استفاده از کدهای درون صورت تمرین و اضافه کردن دو print صحت عملکرد argmax را می‌سنجیم، همانطور که در تصاویر زیر قابل مشاهده است، تابع مذکور به درستی کار می‌کند.

شکل ‏1‑3: صحت عملکرد argmax

سپس لازم است کلاسی بنویسیم به نام GreedyAgent، بدین منظور به الگوریتم Greedy که در کتاب به آن اشاره شده مراجعه می‌کنیم، ضمن اینکه در نوشتن کلاس باید حواسمان باشد از توابعی استفاده کنیم یا درواقع نام توابع به گونه‌ای باشد که با کلاس‌های فراخوانی شده، مطابقت داشته باشد، در صورت تفاوت ارور پیش می‌آید، که یا باید کلاس‌های از پیش نوشته را تغییر دهیم یا نام توابع را اصلاح کنیم، که ما مورد دوم را انجام دادیم و از همان action\_step استفاده کردیم.

A screenshot of a computer

Description automatically generatedبرای اینکه مشکلی پیش نیاید، دایرکتوری کد باید به صورت زیر باشد:

شکل ‏1‑4: دایرکتوری کد نویسی

A screenshot of a computer

Description automatically generatedالگوریتم Greedy مطابق توضیحات کتاب به صورت زیر است:

شکل ‏1‑5: الگوریتم Greedy

A screenshot of a computer

Description automatically generatedبنابراین ابتدا در کد، مقدار N را بروز می‌کنیم، سپس با توجه به رابطه‌ی موجود در شکل 5-1 برای بروز کردن Q، درون کد Q را به روز می‌کنیم، در نهایت براساس خروجی تابع argmax اکشن جدید را انتخاب می‌کنیم، و کلاس ما این اکشن را برمی‌گرداند.

شکل ‏1‑6: کد کلاس GreedyAgent

A screenshot of a computer

Description automatically generatedبا استفاده از کد موجود در صورت تمرین، نمودار را رسم می‌کنیم. توجه: مقدار بهینه با توجه به توضیحات کتاب 1.55 است.

شکل ‏1‑7: خروجی GreedyAgent

همانطور که مشاهده می‌گردد، مطابق انتظار، در حالتی که Agent صرفا حریصانه عمل کند، در کمتر از 40 درصد مواقع عمل بهینه را انتخاب خواهد کرد و به مقدار ارزش بهینه که برابر 1.55 است نخواهد رسید و در حالت زیر بهینه رفتار خواهد کرد.

## پیاده سازی EpsilonGreedyAgent

A screenshot of a computer

Description automatically generatedدر این بخش، روش EpsilonGreedy را پیاده کردیم. کد کلاس آن مشابه قبل است، با این تفاوت که، عددی رندوم تولید می‌کنیم، اگر این عدد کمتر از مقدار epsilon باشد، Agent به جست و جو می‌پردازد و اگر بیشتر از epsilon شود، greedy عمل می‌کند. کد نوشته شده، مطابق زیر است:

شکل ‏1‑8: کد بخش EpsilonGreedyAgent

با استفاده از کد موجود در صورت تمرین، نمودار زیر رسم گردید.

A screenshot of a computer

Description automatically generatedدر این میزان اجرا، همانطور که مشاهده می‌گردد، مقدار epsilon برابر با 0.1 نسبت به سایرین وضعیت بهتری دارد، epsilon برابر با صفر همان عمل صرفا greedy است که گفتیم حالت زیر بهینه خواهد داشت. انتظار می‌رود، در طولانی مدت حالت epsilon برابر با 0.01 از حالت epsilon برابر با 0.1 جلو بزند و به مقدار 1.55 بهینه برسد، چون احتمال انتخاب اکشن بهینه در آن به 99 درصد می‌رسد. از آنجایی که این احتمال برای epsilon برابر با 0.1 نیز 90 درصد است و انتظار می‌رود این نمودار نیز به 1.55 برسد، هرچه epsilon افزایش بیابد، نیز ممکن است به حالت زیر بهینه برویم، این موضوع برای epsilon برابر 0.4 نیز مشهود است، در این حالت احتمال جست و جو افزایش یافته است که این موضوع باعث می‌شود، احتمال انتخاب عمل بهینه کمتر از حالات دیگر بشود.

شکل ‏1‑9: مقایسه‌ی epsilon های مختلف

نمودار بعدی نیز در ادامه در شکل 10-1 موجود است.

A screen shot of a computer

Description automatically generatedتفاوت دو اجرا می‌تواند به دلیل تصادفی بودن محیط باشد، همه چیز به این بستگی دارد که عامل، به صورت تصادفی کدام عمل را برای آغاز انتخاب کند، و چه زمانی به صورت تصادفی شروع به جست و جو بکند، این موارد می‌تواند باعث ایجاد تفاوت بشود. زمانی که محیط نیز پاداش با توزیع احتمال خاص مثلا گاوسی می‌دهد، انتخاب یک عمل مشابه نیز می‌تواند منجر به نتایج متفاوت شود، ممکن است یکبار به ازای یک عمل مشابه پاداش بسیاری دریافت کنیم، یا اینکه پاداش کمتری دریافت شود و همین موضوع فرایند یادگیری را کند کند.

شکل ‏1‑10: مقایسه‌ی دو اجرای متفاوت

## پیاده سازی EpsilonGreedyAgentConstantStepsize

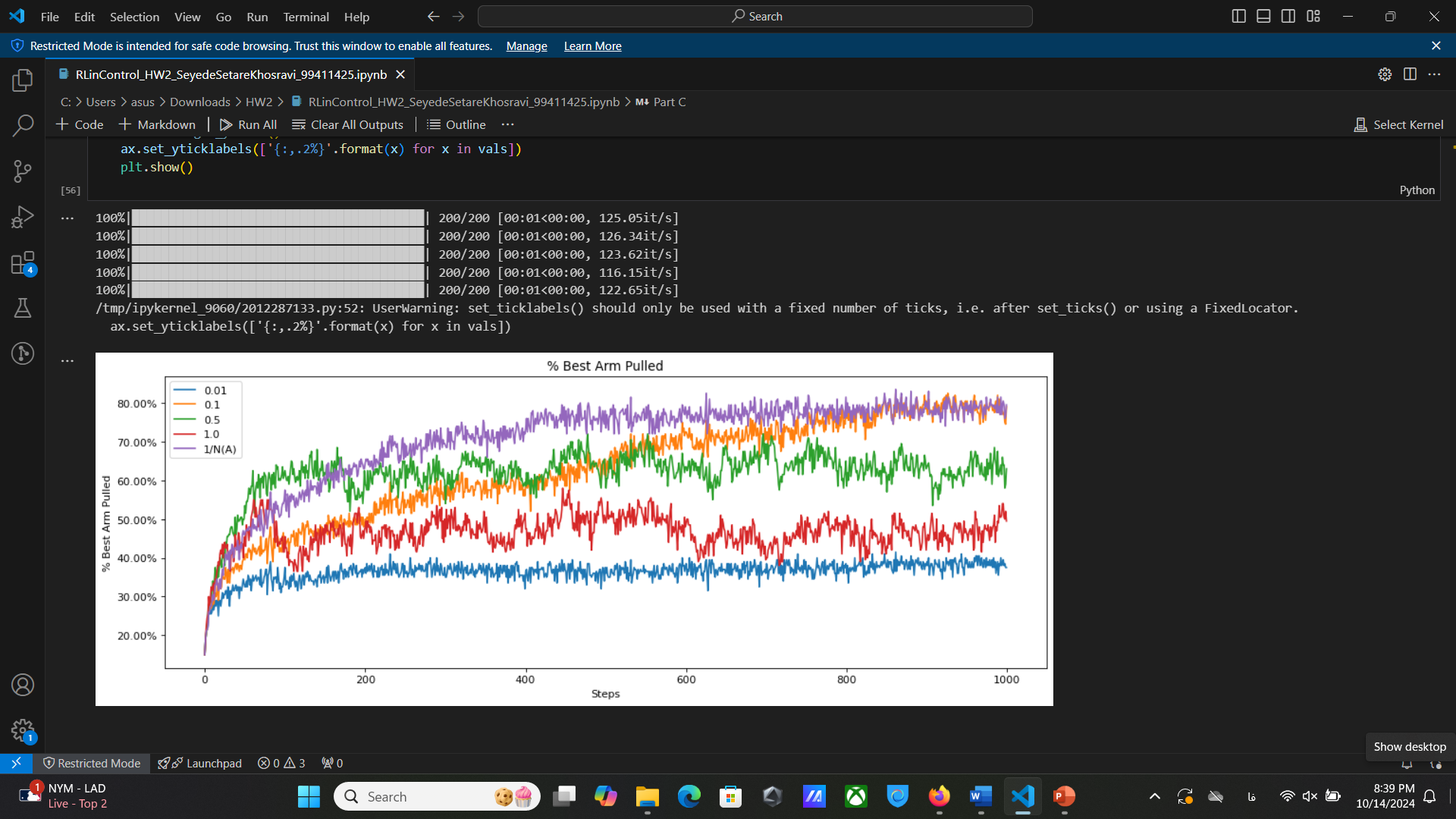
در این قسمت صرفا بجای استفاده از step size متغیر، از خود پارامتر step\_size که در کلاس والد وجود داشت استفاده کردیم (با استفاده از ارث بری این ویژگی به کلاسی که نوشتیم منتقل شده است)

کدی که نوشته شد در شکل 11-1 موجود است.

A screenshot of a computer

Description automatically generated

شکل ‏1‑11: کد FixedStepSize

خروجی کد رسم نمودار که در صورت تمرین آمده است نیز به صورت زیر است:

شکل ‏1‑12: نمودار FixedStepSize

همانطور که مشاهده می‌گردد، بهترین عملکرد برای حالتی است که مشابه EpsilonGreedy عمل می‌کنیم و اندازه گام ثابت نیست. اندازه گام متغیر باعث می‌شود که در ابتدا جست و جوی بیشتری داشته باشیم، فرصت برای میانگین‌گیری باشد و همگرایی نیز براساس این گام تضمین می‌شود، بعلاوه اینکه تاثیر بایاس و شرایط اولیه نیز به زودی از بین می‌رود.

اندازه گام 0.1 نیز می‌تواند عملکرد خوبی را از خود نشان بدهد، اما مشکلی که دارد این است که در این حالت یادگیری کندتر می‌شود، درواقع به نوعی این گام نرخ یادگیری است و هرچقدر کمتر باشد، می‌تواند ما را به سمت همگرایی سوق دهد اما روند یادگیری را کندتر می‌کند، با افزایش مقدار گام به 0.5 و 1، ممکن است سیستم در شرایط زیر بهینه گیر بیفتد، در حالتی که گام برابر با 1 است، مطابق رابطه‌ی موجود برای بروز رسانی ارزش، به طور کلی تخمین با پاداش جدید جایگزین می‌شود که این موضوع مطلوب نیست و باعث نوسان می‌گردد.

در حالتی که اندازه گام را خیلی کوچک انتخاب کنیم، بروز شدن ارزش بسیار کند می‌شود، و زمان زیادی لازم است تا ارزش به صورت قابل توجهی بروز شود و همگرایی بشدت کند می‌گردد.

## پیاده سازی UCBAgent

در این قسمت با استفاده از کلاسی که برای GreedyAgent نوشتیم، فقط با اضافه کردن ترم عدم قطعیت، آن را به UCB تبدیل کردیم، با توجه به اینکه رسم نمودار آن با مشکل مواجه گردید، از پیاده سازی دیگری استفاده شد، در این پیاده سازی منطق مطابق کلاس قبلی است که برای UCBAgent و EpsilonGreedyAgent نوشتیم و تفاوتی در کد موجود نیست فقط رسم نمودار این دو در کنار هم به درستی انجام می‌گردد.

A screenshot of a computer

Description automatically generatedکد اولیه به صورت زیر است:

شکل ‏1‑13: کد UCB

A screenshot of a computer

Description automatically generatedپیاده سازی صحیح آن نیز به صورت زیر است:

شکل ‏1‑14: بخش اول کد

A screenshot of a computer

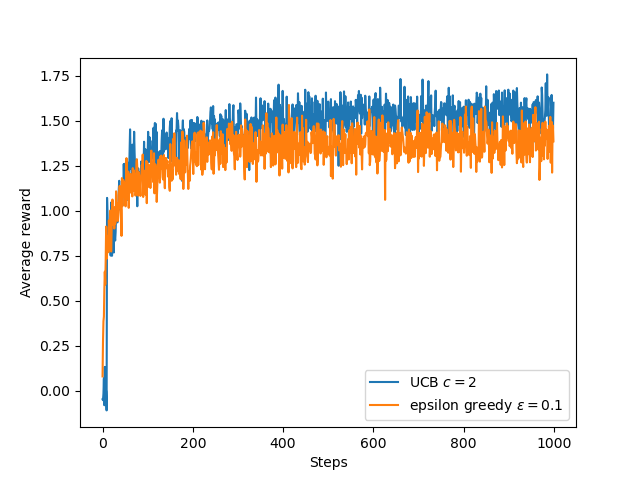
Description automatically generatedA screenshot of a computer

Description automatically generatedدر بخش دوم کد همانطور که می‌بینید، هم بروز رسانی مطابق الگوریتم موجود است هم به صورت sample average که ما مطابق الگوریتمی که اشاره کردیم، رفتار می‌کنیم و sample average را در ابتدا false کردیم. سایر بخش‌های کد نیز مشابه کلاس‌هایی است که نوشتیم.

شکل ‏1‑15: بخش آخر کد UCB

شکل ‏1‑16: بخش دوم کد UCB

در نهایت نیز اجرا و رسم نمودار است.

خروجی نیز در تعداد اجرای 200 به صورت زیر است:

شکل ‏1‑17: خروجی UCB و EpsilonGreedy

و پس از 2000 اجرا نیز خروجی در شکل 18-1 قابل مشاهده است.

A graph of steps and steps

Description automatically generated

شکل ‏1‑18: خروجی UCB و EpsilonGreedy پس از 2000 اجرا

همانطور که مشاهده می‌گردد، عملکرد UCB از EpsilonGreedy بهتر است، خصوصا زمانی که دفعات اجرا بالاتر می‌رود، UCB به بهترین شکل، مسبت به روش‌های دیگری میان بهره برداری و جست و جو تعادل را برقرار می‌کند، که این موضوع به علت افزودن ترم عدم قطعیت می‌باشد. در UCB ابتدا، تمامی عمل‌ها انتخاب می‌شوند، در ادامه نیز اعمال با توجه به تعداد دفعاتی که انتخاب شده‌اند و توجه به گذر زمان، شانس انتخاب مجدد دارند حتی اگر بهینه نباشند، این روش تضمین می‌دهد که همه‌ی عمل‌ها حتی اعمال غیر بهینه با گذر زمان انتخاب خواهند شد، و در نهایت نیز به حالت بهینه همگرا می‌شویم، این در حالی است که در EpsilonGreedy بحث عدم قطعیت مطرح نیست و جست و جو با احتمال یکسان و کاملا تصادفی میان اعمال انجام می‌پذیرد. UCB به دلیل انجام اکتشاف حتی پس از رسیدن به حالت بهینه، باعث ایجاد عملکرد بهتری می‌شود، زیرا تضمین می‌دهد که اطلاعات کافی از تمام اعمال را جمع‌آوری می‌کند. اگر پاداش‌ها نیز تصادفی باشند، به دلیل اینکه UCB براساس عدم قطعیت نیز عمل می‌کند و میان جست و جو و انتخاب حریصانه تعادل خوبی برقرار می‌کند، و جست و جو را همچنان ادامه می‌دهد، می‌تواند زمانی که محیط تصادفی است نیز بهتر از روش‌های دیگر عمل کند. همانطور که مشاهده می‌گردد با افزایش دفعات اجرا نیز تفاوت میان UCB و Epsilon Greedy مشهود تر است، به این علت که UCB با توجه به عملکرد خاص خود و برقراری تعادل میان explore و exploit به نحوی بهتر از سایر روش‌ها، در دراز مدت اطلاعات مفیدی از محیط کسب می‌کند، و عملکردی بسیار نزدیک به حالت بهینه دارد و عامل با این روش می‌تواند بهینه‌تر عمل کند.

رفرنس‌ها:

کتاب Sutton and Barto

ریپازیتوری‌های:

<https://github.com/imimali/reinforcement-learning-specialization>

<https://github.com/setarekhosravi/reinforcement-learning-an-introduction>