زن زندگی آزادی

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر

گزارش تمرین شماره سوم
درس یادگیری تعاملی
پاییز ۱۴۰۱

نام و نام خانوادگی
سیاوش رزمی
شماره دانشجویی
۸۱۰۱۰۰۳۵۲

فهرست

چکیده ۳
مسئله ۱ -مدل سازی ۴
هدف سوال ۴
سوال ۲ - سوال تئوری ۷
هدف سؤال ۷
سوال۳ - ۲۱
نکات مهم و موارد تحویلی ۳۶
موارد تحویلی ۳۶
منابع ۳۷

چکیده

در این تمرین قصد داریم به بررسی مسائل MDP بپردازیم، در تمرین اول سعی میکنیم مسأله های دنیای واقعی را توسط مفاهیم MDP مدل کنیم و چالش های مدل‌سازی MDP را درک کنیم .

سپس در تمرین دوم روند انجام الگوریتم های Policy iteration وValue Iteration و نحوه همگرایی و رسیدن به سیاست بهینه توسط این دو روش با محاسبه به شکل دستی در یک مثال ساده را بررسی میکنم.

در تمرین سوم نیز به پیاده‌سازی این دو الگوریتم با استفاده از برنامه نویسی پایتون در یک مثال نسبتاً پیچیده‌تر می‌پردازیم و عمل‌کرد الگوریتم ها را در محیط پایتون تحت پارامتر های مختلف بررسی میکنیم.

مسئله ۱ -مدل سازی

هدف سوال

در این بخش قصد داریم دو مسأله از دنیای واقعی را با استفاده از مفاهیم مدل MDP مـدل سـازی و حل و مزایا و معایب این روش مدلسازی را بررسی کنیم.

سؤال یک - در این سؤال قصد داریم یک برنامه ورزشی و رژیم طراحی کنیم که بـا توجـه بـه عوامـل محیط و عمل‌کرد کاربر برنامه‌های متفاوتی را به وی پیشنهاد بدهد.

در ابتدا به توضیح کلی عمل‌کرد این برنامه می‌پردازیم.

در ابتدا کاربر برنامه بایستی ثبت نام کرده و مشخصات فیزیکی خود ماننـد جنسـیت، قـد ، وزن، سـن و.. خود به همراه یک سری سؤال که مربوط به علایق و ترجیح های وی مانند غـذا هـای مـورد علاقـه، روز های هفته که علاقه به ورزش و تمـرین دارد، محیطی کـه در آن معمـولاً ورزش و فعـالیت میکنـد را در برنامه ثبت کند.

با استفاده از این مشخصات میتوان برای هر کاربر یک پرسونا تعریف کرد و با توجـه بـه این پرسـونا یـک سری برنامه اولیه به وی پیشنهاد داد که این کار مشکل Cold start را برای مدل تا حد زیادی بر طـرف خواهد کرد.

حال پس از ثبت نام و با توجه به اطلاعات کاربر میتوان یک برنامه ورزشی و رژیم مناسب با وی را بـه او پیشنهاد داد.

برنامه به شکل پیوسته در طول زمان از کاربر به اشـکال مختلـف بـازخورد دریـافت میکنـد و برنامه‌هـای پیشنهادی را توسط این اطلاعات در طول زمان نسبت به علایق کاربر بهینه میکند.

بازخورد های دریافتی از کاربر میتواند اشکال گوناگونی داشته باشد به طور مثال:

۱) بعد انجام هر تمرین از کاربر بازخورد دریافت می‌شود تا میزان تناسب برنامـه بـا علایـق وی بررسی شود این بازخورد میتوان به شکل میزان سختی برنامه از نظر کاربر یا میزان امتیـازی که وی به این تمرین میدهد باشد.

۲) در هنگام تمرین با استفاده از سنسور های موجود بـه طـور مثـال مچ بنـد هوشـمند یـا ساعت هوشمند یا دوربین تلفن همراه از علایم حیاطی کاربر ماننـد ضـربان قلب، مـیزان کـالُری مصرفی و یا حالت صورت وی هنگام فعالیت و... بازخورد دریافت می‌شود.

۳) در حین تمرین امکان پرش از روی یک تمرین خاص و یا اتمام تمرین در میانه و یـا کم کردن تعداد باری که بایستی انجام شود و همچنین افزایش میزان هر تمرین یا تکـرار آن وجـود خواهد داشت که در صورت اقدام کاربر به این کار این اطلاعات نیز ذخیره می‌شود.

۴) همچنین متعهد بودن کاربر به انجام تمرین در طول هفته نیز میتواند پارامتر مفیدی در میزان تناسب تمرین با شرایط وی باشد به طور مثال اگر کاربر تعیین کـرده اسـت کـه در طـول هفته روز های زوج به تمرین بپردازد و بعد از پیشـنهاد یـک تمـرین چنـد روز از انجـام تمـرین خودداری کند این یک بازخورد به شدت منفی محاسبه می‌شود و حتی میتـوان در اولین مرتبـه که وی دوباره به نرم‌افزار وارد شده دلیل عدم انجام تمرین را از وی جویا شد.

حال به مدل سازی مدل MDP با استفاده از این اطلاعات می‌پردازیم:

۱- Agent: عامل در این مدل سیستم توصیه گر است.

۲- Action: حذف یا اضافه کردن و یا تغییر تعداد هرکدام از تمرین‌ها از برنامه پیشنهادی است.

۳- State: برنامه های پیشنهاد شده به کاربر است.

۴- Environment: کاربر برنامه که با استفاده از این برنامه‌ها تمرین میکند.

۵- Reward: تمامی بازخورد هایی که در بخش قبلی تعریف شده میتواند در محاسبه پاداش لحاظ شود برای تجمیع این بازخورد ها میتوان به اشکال گوناگون عمل کرد یک روش ساده این است که به هرکدام از این فاکتور ها یک وزن بخصوص داده و با میانگین یا جمع وزن دار آن‌ها میزان پاداش کل را بدست آوریم همچنین میتوان چندین فاکتور مختلف را بررسی و در صورت اتفاق نظر همگی آن‌ها آنرا به پاداش اعمال کرد با این کار تا حدی نویز احتمالی در جمع آوری این اطلاعات از آن حذف خواهد شد به طور مثال در صورتی که میزان کالُری سوزانده شده توسط کاربر خیلی زیاد باشد و Facial Expression وی نیز در طول تمرین مثبت باشد و تمرین را نیز به طور کامل و تا انتها اجرا کرده باشد اما امتیازی که وی به این تمرین داده است مقدار ۰ ستاره باشد احتمال زیادی وجود دارد که در انتهای تمرین و به دلیل خستگی زیاد اشتباهاً این امتیاز را وارد کرده است.

مزایا و معایب این مدل نسبت به مدل توصیه گر عادی:

رفتار و علایق کاربر در طول زمانی ممکن است با فرکانس بسیار بالایی تغییر کند که مدل های توصیه گر عادی ممکن است به سرعت نتواند با این تغییرات سازگار شود اما مدل های یادگیری تقویتی میتوانند با سرعت بالاتری خود را با شرایط وقف دهند.

مدل های یادگیری تقویتی اما برای آموزش نیاز به دیتا زیادی دارند همچنین مدل های یادگیری تقویتی به طور کلی از مدل های عادی پیچیده‌تر و پارامتر های بیشتر دارند که این مسأله باعث می‌شود حجم محاسبه ما بالاتر رود.

مدل های عادی توصیه‌گر برای هر فرد به طور مجزا آموزش داده نمی‌شوند و برای توصیه برنامه‌ها عمل‌کرد فرد کاربر و بازخورد وی را مدنظر قرار نمی‌دهند.

سؤال دوم امتیازی-

فرض کنیم در ده n نفر آدم زندگی می‌کنند و قرعه کشی به شکل دوره ایی در ده برگزار می‌گردد،اهالی ده از انتخاب همدیگر خبر دارند و طبق اعلان افراد انتخاب می‌کنند که در قرعه کشی بعدی شرکت کنند یا نکنند میزان مبلغ جایزه همیشه ثابت است و احتمال انتخاب شدن هر فرد در قرعه کشی ۱ تقسیم بر تعداد کل افراد شرکت کننده در قرعه کشی باشد، هر یک از افراد اختیار دارد در قرعه کشی شرکت کند یا نکند، بنابرین اکشن ما یک ماتریس n بعدی است که هر کدام از آرایه های آن انتخاب فرد متناظر برای شرکت یا عدم شرکت در قرعه کشی می‌باشد، همچنین هر تعداد افراد شرکت کننده در قرعه کشی بالاتر رود احتمال برنده شدن افراد کمتر و همچنین احتمال سرایت بیماری به آن‌ها بیشتر خواهد شد بنابرین پاداش ما میتواند یک آرایه n بعدی که هر عنصر آن برای هر فرد برنده شدن جایزه و یا گرفتن بیماری باشد و یا مقدار متوسطی از آن باشد، حالت‌های ما بر اساس تعداد افراد مبتلا به بیماری می‌باشد به شکلی که با هربار انتخاب همه افراد ده در مورد اینکه در قرعه کشی شرکت کنند و یا نکنند تعداد مبتلایان تغییر کرده و در نتیجه احتمال مبتلا شدن و احتمال برنده شدن برای همه افراد تغییر می‌کند، محیط ما قرعه کشی دوره ایی است.

سوال ۲ - سوال تئوری

هدف سؤال

در این تمرین روند انجام الگـوریتم هـای Policy iteration و Value Iteration و نحـوه همگـرایی و رسیدن به سیاست بهینه توسط این دو روش با محاسبه به شکل دسـتی در یـک مثـال سـاده را بررسـی خواهیم کرد در ابتدا به تأثیر مقدار discount factor در سیاست بدست آمـده توسـط این الگـوریتم هـا می‌پردازیم سپس اهمیت ارزش‌های حالت هارا در همگرایی بررسی میکنیم.

سؤال ۱-

محیط داده شده توسط سؤال یک زمین با ابعاد ۳ در ۴ سلول می‌باشد که برای سهولت کار خانه‌های این زمین را از خانه شروع تا خانه هدف از چپ به راست عدد گذاری میکنیم:

0	1	2 Hell	3
4 Obstacle	5	6	7
8	9	10	11 Goal

شکل ۱-۲: محیط مسأله و نحوه شماری گذاری

برای حل مسأله به شکل policy iteration از الگوریتم زیر استفاده شده است:

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

شکل ۲-۲: شبه کد الگوریتم Policy iteration

ارزش تمامی خانه در ابتدا صفر و مقدار theta نیز برابر با ۰.۱ در نظر گفته شده است.
سیاست اولیه نیز به ازای تمامی خانه‌ها برابر با عمل (بالا) انتخاب شد.

عامل با انتخاب اکشن بالا در خانه ۰ با احتمال ۱ به خانه ۰ بر می‌گردد(احتمال انتقال برای مابقی خانه‌ها صفر است) و پاداش صفر دریافت میکند با توجه به اینکه discount factor صفر است بنابرین در هر انتقال صرفاً پاداش دریافت شده را در ارزش آن خانه لحاظ میکنیم.

$V(0) = P(s' = 0,r|s=0,up)\ [Reward + gamma * V(s')] = 1 * [0 + 0 * 0] = 0$

$V(1) = P(s'=1,r|s=1,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

با رسیدن به خانه دوم با توجه به اینکه Terminal State است از به روز رسانی آن خودداری میکنیم

$V(2) = Terminal$

$V(3) = P(s'=3,r|s=3,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

$V(4) = Terminal$

$V(5) = P(s'=1,r|s=5,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

$V(6) = P(s'=2,r|s=6,up)[Reward + gamma * V(s')] = 1* [-10+0*0] = -10$

$V(7) = P(s'=3,r|s=17,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

$V(8) = P(s'=4,r|s=8,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

$V(9) = P(s'=5,r|s=9,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

$V(10) = P(s'=6,r|s=10,up)[Reward + gamma * V(s')] = 1* [0+0*0] = 0$

$V(11) = Terminal$

پس از اتمام یک بار حلقه بر روی تمام حالت‌ها یک بار دیگر این گام را تکرار کرده و در صورت عدم تغییر مقادیر بیش از مقدار theta به مرحله Improvement می رویم:

$V(0) = 0$

$V(1) = 0$

$V(2) = Terminal$

$V(3) = 0$

$V(4) = Terminal$

$V(5) = 0$

$V(6) = -10$

$V(7) = 0$

$V(8) = 0$

$V(9) = 0$

$V(10) = 0$

V(11) = Terminal

به دلیل عدم تغییر مقادیر به مرحله Improvement میرویم:

Policy Improvement:

برای محاسبه بهترین اکشن در هر حالت ابتدا مقدار Q برای هر اکشن در هر حالت را محاسبه کرده سپس اکشن با بیشتر مقدار Q را انتخاب میکنیم (در صورتی که مقادیر مساوی به دست آمد اولین اکشن از سمت چپ را به عنوان اکشن بهینه انتخاب میکنیم):

P(0) = argmax (P(s'=0,r|s=0,up)[0+ 0*V(0)], P(s'=0,r|s=0,down)[0+ 0*V(0)], P(s'=0,r|s=0,left)[0+ 0*V(0)], P(s'=1,r|s=0,right)[0+ 0*V(1)]) = argmax(0,0,0,0) =up

با توجه به اینکه تمام مقادیر صفر هستند اولین اکشن را به عنوان اکشن بهینه انتخاب میکنیم:

P(1) = argmax (P(s'=1,r|s=1,up)[0+ 0*V(0)], P(s'=5,r|s=1,down)[0+ 0*V(5)], P(s'=0,r|s=1,left)[0+ 0*V(0)], P(s'=2,r|s=1right)[-10+ 0*V(2)]) = argmax(0,0,0,-10) = up

P(2) = Terminal

P(3) = argmax (P(s'=3,r|s=3,up)[0+ 0*V(3)], P(s'=8,r|s=3,down)[0+ 0*V(8)], P(s'=2,r|s=1,left)[-10+ 0*V(2)], P(s'=3,r|s=3,right)[0+ 0*V(3)]) = argmax(0,0,0,-10) = up

P(4) = Terminal

P(5) = argmax (P(s'=1,r|s=5,up)[0+ 0*V(1)], P(s'=9,r|s=5,down)[0+ 0*V(9)], P(s'=5,r|s=5,left)[0+ 0*V(5)], P(s'=6,r|s=5,right)[0+ 0*V(6)]) = argmax(0,0,0,0) = up

P(6) = argmax (P(s'=2,r|s=6,up)[-10+ 0*V(2)], P(s'=10,r|s=6,down)[0+ 0*V(10)], P(s'=5,r|s=6,left)[0+ 0*V(5)], P(s'=7,r|s=6,right)[0+ 0*V(7)]) = argmax(-10,0,0,0) =down

P(7) = argmax (P(s'=3,r|s=7,up)[0+ 0*V(3)], P(s'=11,r|s=7,down)[11+ 0*V(11)], P(s'=6,r|s=7,left)[0+ 0*V(6)], P(s'=8,r|s=7,right)[0+ 0*V(8)]) = argmax(0,10,0,0) =down

P(8) = argmax (P(s'=8,r|s=8,up)[0+ 0*V(8)], P(s'=8,r|s=8,down)[0+ 0*V(8)], P(s'=8,r|s=8,left)[0+ 0*V(8)], P(s'=9,r|s=7,right)[0+ 0*V(9)]) = argmax(0,0,0,0) = up

P(9) = argmax (P(s'=5,r|s=9,up)[0+ 0*V(5)], P(s'=9,r|s=9,down)[0+ 0*V(9)], P(s'=8,r|s=9,left)[0+ 0*V(8)], P(s'=10,r|s=9,right)[0+ 0*V(10)]) = argmax(0,0,0,0) = up

P(10) = argmax (P(s'=6,r|s=10,up)[0+ 0*V(6)], P(s'=10,r|s=10,down)[0+ 0*V(10)], P(s'=9,r|s=10,left)[0+ 0*V(9)], P(s'=11,r|s=10,right)[10+ 0*V(11)]) = argmax(0,0,0,10) = right

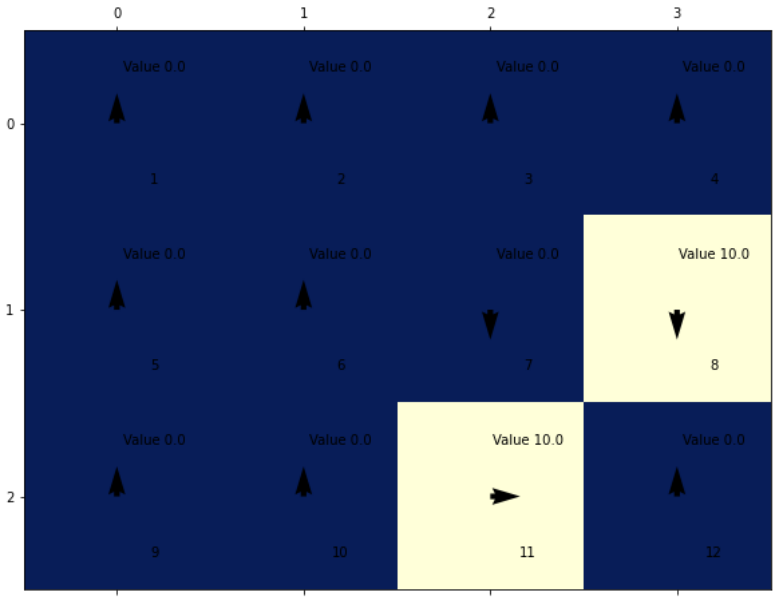
با توجه به اینکه سیاست بعد از مرحله Improvement تغییر کرده است دوباره به مرحله Evaluation باز میگردیم و مقادیر را محاسبه میکنیم:

برای راحتی کار مابقی مراحل به شکل خلاصه در جدول زیر آورده شده است:

جدول ۱-۲: مراحل به روز رسانی حالت و سیاست در الگوریتم Policy iteration با مقدار گاما برابر با صفر، ستون V نشانگر ارزش حالت‌ها در سیاست فعلی و مقدار P نشان دهنده اکشن انتخابی توسط سیاست است، اکشن ها با حرف اول لاتین مشخص شده اند، رنگ قرمز نشانگر مقدار اولیه،رنگ آبی مشخص کننده مقدار به روز رسانی شده و رنگ سبز مقادیر بهینه هستند.

	Intial Value		Iteration=1		Iteration=2		Iteration =3		iteration=4		iteration=5	
State	V	P	V	P	V	P	V	P	V	P	V	P
1	0	U	0	U	0	U	0	U	0	U	0	U
2	0	U	0	U	0	U	0	U	0	U	0	U
3	0	U	0	U	0	U	0	U	0	U	0	U
4	0	U	0	U	0	U	0	U	0	U	0	U
5	0	U	0	U	0	U	0	U	0	U	0	U
6	0	U	0	D	0	D	0	D	0	D	0	D
7	0	U	10	D	10	D	10	D	10	D	10	D
8	0	U	0	U	0	U	0	U	0	U	0	U
9	0	U	0	U	0	U	0	U	0	U	0	U
10	0	U	10	R	10	R	10	R	10	R	10	R
11	0	U	0	U	0	U	0	U	0	U	0	U

در مرتبه دوم با توجه به اینکه سیاست بهینه تغییر نکرده است الگوریتم پایان می‌یابد.



شکل ۳-۲: سیاست و ارزش نهایی الگوریتم Policy Iteration با گاما برابر صفر

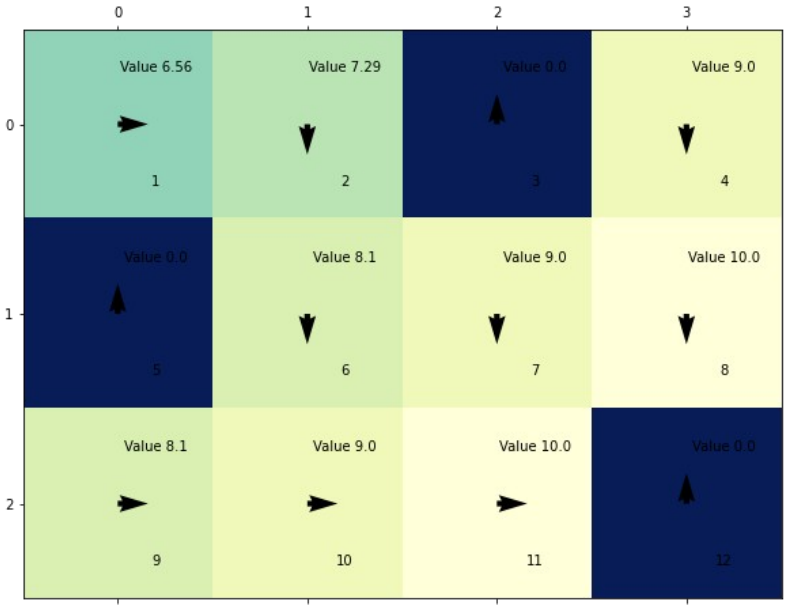
تحلیل: با توجه به اینکه Discount factor صفر است ارزش تمامی خانه‌های دور از خانه هدف صفر است تنها خانه‌هایی که چسبیده به خانه هدف هستند ارزش مخالف صفر داشته و میتوانند خانه هدف را پیدا کنند و مابقی خانه‌ها سیاست بهینه را پیدا نمی‌کنند البته خانه‌هایی هم که نزدیک به خانه Hell هستند نیز با توجه به اینکه پاداش رفتن در این خانه ۱۰ – است بعد از یک مرتبه اجرای الگوریتم هیچ‌وقت در این خانه نمی‌رود، بنابرین میتوان گفت هنگامی که مقدار Discount factor صفر است عامل به شکل myopic عمل میکند و در خانه‌هایی به جز خانه‌های چسبیده به هدف ممکن است هیچ‌گاه به خانه هدف نرسد.

سؤال ۲- طبق الگوریتم و توضیحات مشابه سوال قبل با Discount factor برابر با ۰.۹ مسأله را حل میکنیم:

با توجه به سنگین بودن محاسبات تمامی مقادیر در مرتبه اول محاسبه شدند اما بـرای محاسـبه مراحـل بعدی از یک کد پایتون استفاده شده است که کد به همراه گزارش ارسال می‌گردد.

جدول ۲-۲: مراحل به روز رسانی حالت و سیاست در الگوریتم Policy iteration با مقدار گاما برابر با ۰.۹،ستون V نشانگر ارزش حالت‌ها در سیاست فعلی و مقدار P نشان دهنده اکشن انتخابی توسط سیاست است، اکشن ها با حرف اول لاتین مشخص شده اند، رنگ قرمز نشانگر مقدار اولیه،رنگ آبی مشخص کننده مقدار به روز رسانی شده و رنگ سبز مقادیر بهینه هستند.

	Initial Value		Iteration=1		Iteration =2		Iteration =3		Iteration=4		Itertaion=5	
State	V	P	V	P	V	P	V	P	V	P	V	P
0	0	U	0	U	0	U	0	U	0	R	6.56	R
1	0	U	0	U	0	U	0	D	7.29	D	7.29	D
2	0	U	0	U	0	U	0	U	0	U	0	U
3	0	U	0	U	0	D	9	D	9	D	9	D
4	0	U	0	U	0	U	0	U	0	U	0	U
5	0	U	0	U	0	R	8.1	D	8.1	D	8.1	D
6	0	U	10-	R	9	D	9	D	9	D	9	D
7	0	U	0	D	10	D	10	D	10	D	10	D
8	0	U	0	U	0	U	0	R	8.1	R	8.1	R
9	0	U	0	U	0	R	9	R	9	R	9	R
10	0	U	9-	R	10	R	10	R	10	R	10	R
11	0	U	0	U	0	U	0	U	0	U	0	U



شکل ۴-۲: نتیجه نهایی الگوریتم با استفاده الگوریتم Policy Iteration با گاما برابر با ۰.۹

تحلیل: با توجه به مراحل به روز رسانی توسط الگوریتم مشخص است که تمامی خانه‌هایی که غیر ترمینال هستند در به روز رسانی پنجم سیاست بهینه (یعنی سیاستی که عامل را به خانه هدف می‌رساند) را پیدا میکنند، با توجه به اینکه مقدار هزینه حرکت برای عامل صفر است بنابرین در الگوریتم مسافت طی شده توسط عامل در نظر گرفته نمی‌شود که ممکن است باعث شود مسیر بهینه پیدا شده لزوماً کوتاه ترین مسیر نباشد، همچنین اگر مدت زمان همگرایی به سیاست بهینه را برای تمام حالت‌ها در نظر بگیریم حالت‌هایی که بیشترین فاصله را از خانه هدف دارند دیرتر از همگی به سیاست بهینه رسیده اند، هر چه مقدار Discount factor را نزدیک‌تر به ۱ بگیریم برای تعیین سیاست بهینه خانه‌های دورتر بیشتر در نظر گرفته می‌شوند در نتیجه در این سؤال به نظر می‌رسد اگر مقدار گاما را افزایش دهیم در تعداد کمتری از اجرای الگوریتم به سیاست بهینه دست یابیم، در بررسی ارزش حالت‌ها در سیاست بهینه نیز به نظر می‌رسد هرچه خانه به خانه هدف نزدیک‌تر باشد و در تعداد حرکت کمتری به خانه هدف برسد ارزش خانه در این سیاست بیشتر خواهد بود که کاملاً منطقی است با توجه به اینکه مقدار ارزش خانه از Discounted Return اکشن بهینه بدست می‌آید هر چه خانه نزدیک‌تر به خانه هدف باشد میزان پاداش رسیدن به خانه هدف در مقدار Discounted factor کمتری ضرب خواهد شد.

سؤال ۳ -

برای محاسبه مقادیر در این قسمت از الگوریتم Value Iteration به شکل زیر استفاده میکنیم:

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

شکل ۵-۲: شبه کد الگوریتم Value Iteration

در این الگوریتم برای حساب کردن ارزش هر حالت بر خلاف الگوریتم Policy Iteration که از مقدار متوسط مقادیر Discounted Return به ازای سیاست ضرب در احتمال Transition استفاده می‌کردیم از ماکزیمم مقادیر به ازای تمامی اکشن استفاده میکنیم و این عمل را تا زمانی که تغییرات ارزش‌ها از یک مقدار theta کوچکتر شود ادامه می‌دهیم در انتها با بدست آوردن مقادیر بهینه ارزش‌ها یک سیاست کاملاً حریصانه به ازای این مقادیر به دست می‌آوریم.

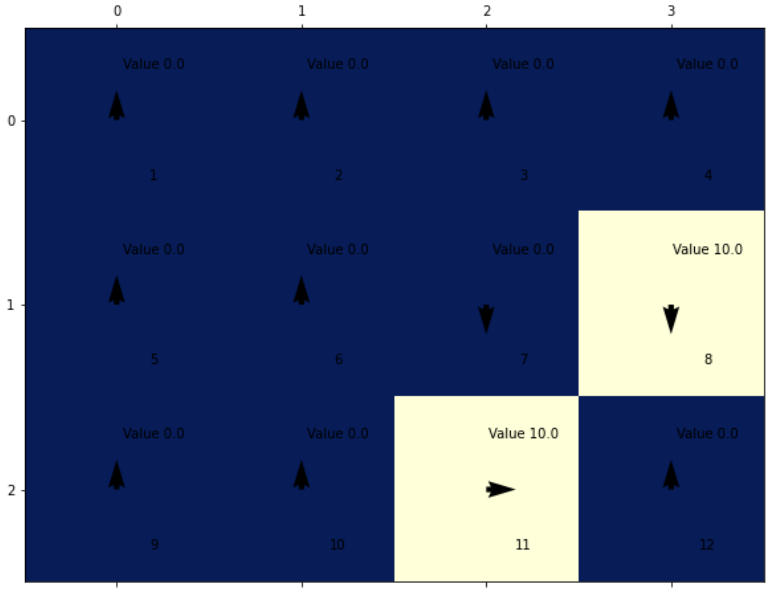
ارزش تمامی خانه در ابتدا صفر و مقدار theta نیز برابر با ۰.۱ و مقدار گاما برابر با ۰.۹ در نظر گفته شـــده است.

سیاست اولیه نیز به ازای تمامی خانه‌ها برابر با عمل (بالا) انتخاب شد.

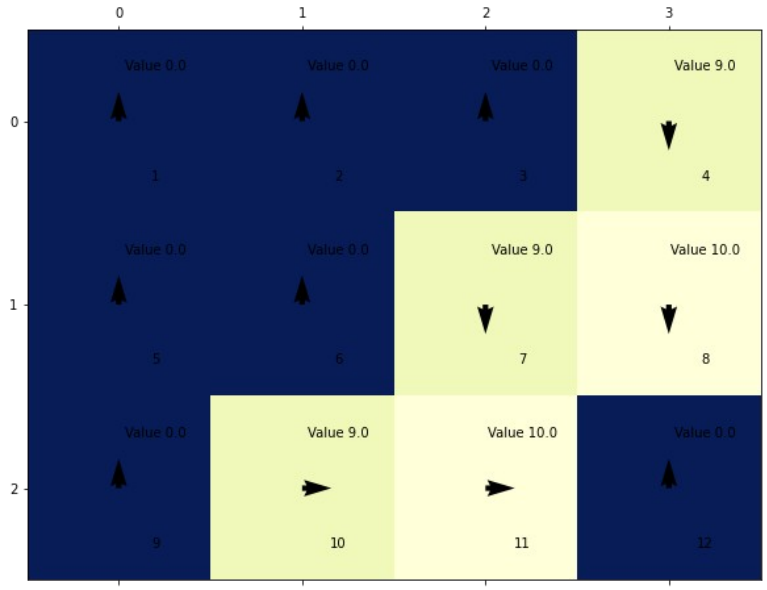
جدول ۳-۲: مراحل به روز رسانی در الگوریتم Value Iteration

State	Initial Values	Iter=1	Iter=2	Iter=3	Iter=4	Iter=5	Iter=6	Optimal Policy
0	0	0	0	0	0	6.561	6.561	R
1	0	0	0	0	7.29	7.29	7.29	D
2	0	0	0	0	0	0	0	U
3	0	0	9	9	9	9	9	D
4	0	0	0	0	0	0	0	U
5	0	0	0	8.1	8.1	8.1	8.1	D
6	0	0	9	9	9	9	9	D
7	0	10	10	10	10	10	10	D
8	0	0	0	8.1	8.1	8.1	8.1	R
9	0	0	9	9	9	9	9	R
10	0	10	10	10	10	10	10	R
11	0	0	0	0	0	0	0	U

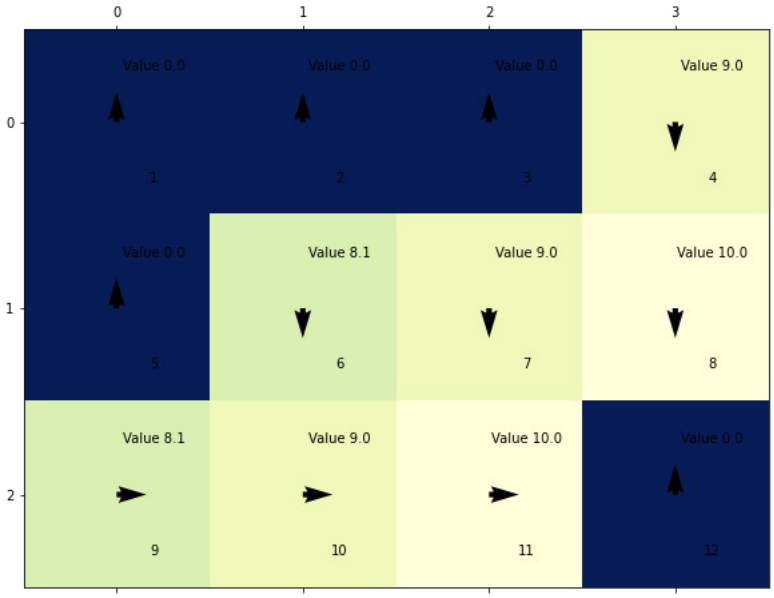
حال به بررسی مقادیر به دست آمده و سیاست بهینه در هر مرتبه اجرای الگوریتم می‌پردازیم:



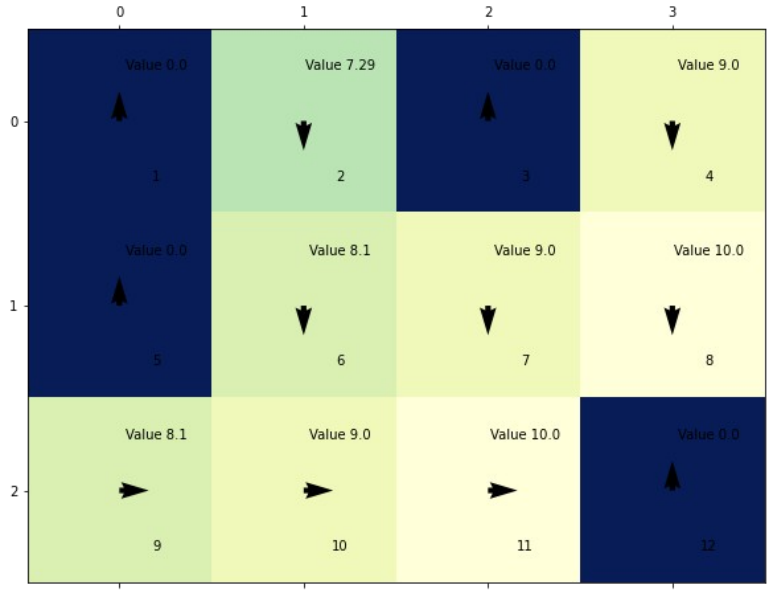
شکل ۶-۲: مقادیر ارزش و سیاست بهینه در اولین مرتبه اجرای الگوریتم Value Iteration



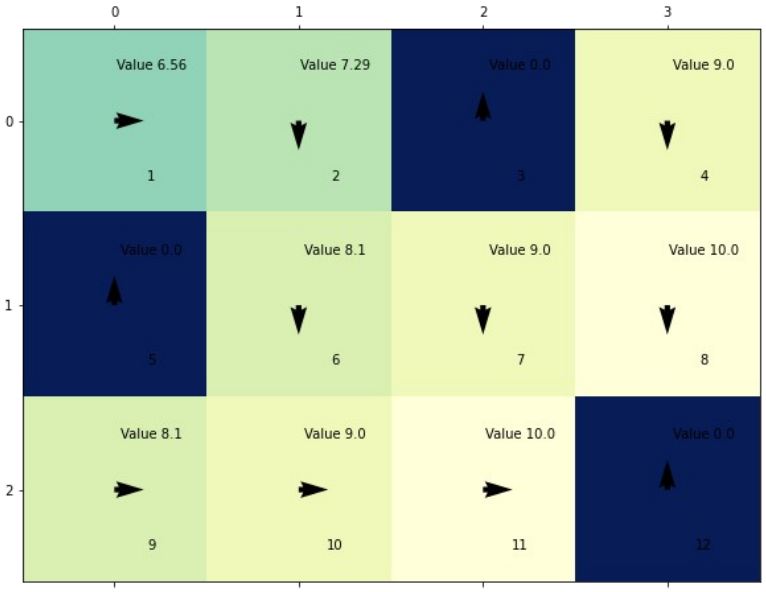
شکل ۷-۲: مقادیر ارزش و سیاست بهینه در دومین مرتبه اجرای الگوریتم Value Iteration



شکل ۸-۲:مقادیر ارزش و سیاست بهینه در سومین مرتبه اجرای الگوریتم Value Iteration



شکل ۹-۲: مقادیر ارزش و سیاست بهینه در چهارمین مرتبه اجرای الگوریتم Value Iteration



شکل ۱۰-۲: سیاست نهایی بدست آمده توسط الگوریتم Value Iterationدر مرتبه پنجم

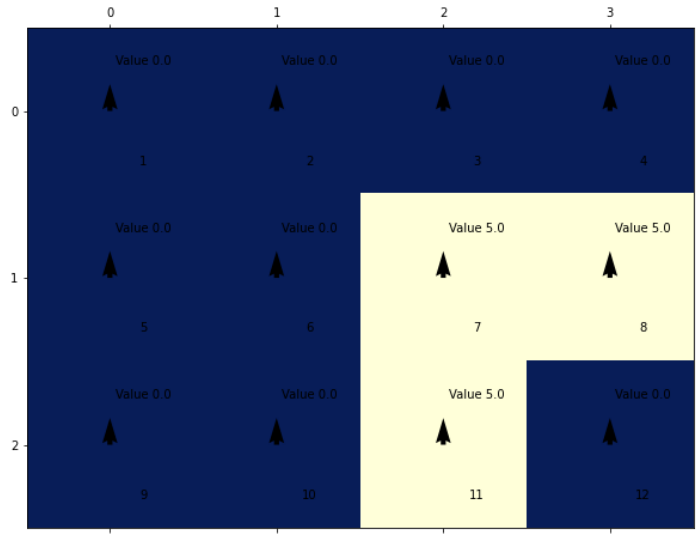
مقادیر نهایی به دست آمده کاملاً شبیه به مقادیر به دست آمده در الگوریتم Policy Iteration است، مفهوم مقادیر ارزش مانند الگوریتم قبلی با فاصله آن خانه تا خانه هدف رابطه عکس دارد به این معنا که هرچه طبق سیاست به دست آمده حرکات کمتری برای رسیدن از آن خانه تا خانه هدف وجود داشته باشد ارزش خانه بیشتر خواهد بود، در صورتی که در میانه اجرای الگوریتم و در هنگام به روز رسانی ارزش‌ها سیاست را به روز کنیم میتوان مشاهده کرد برای خانه‌های نزدیک به خانه هدف این سیاست، سیاست بهینه است اما برای خانه‌های دور از خانه هدف ممکن است سیاست ما بهینه نباشد و مارا به خانه هدف نرساند همچنین قابل مشاهده است که خانه‌های نزدیک به خانه هدف سریع‌تر از بقیه به سیاست بهینه دست پیدا میکنند و مابقی خانه‌ها پس از اینکه خانه‌های مذکور به سیاست بهینه رسیدند شروع به پیدا کردن اکشن مناسب می‌کنند.

سؤال ۴-

برای امتحان کردن تأثیر مقادیر اولیه Value در سرعت همگرایی الگوریتم چندین نمونه مختلف تغییر مقادیر امتحان شد:

۱- زیاد کردن ارزش خانه‌های چسبیده به خانه هدف:

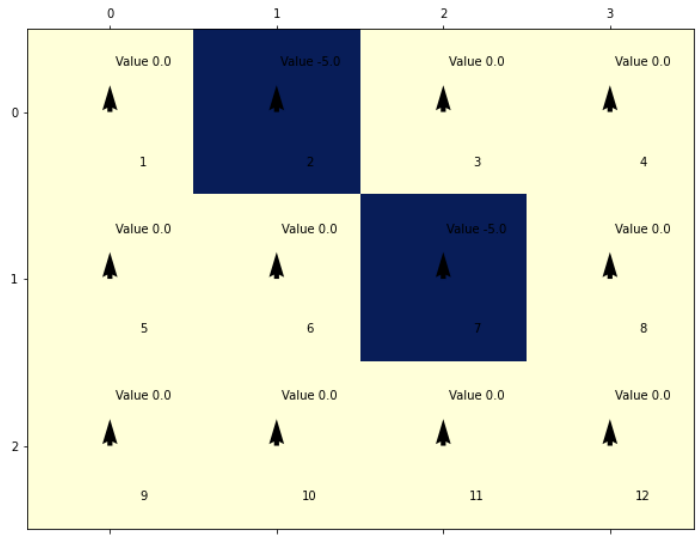
به خانه‌های ۷، ۸ و ۱۱ ارزش‌های ۵ داده شد و مابقی خانه‌ها ارزش صفر مشابه دفعه‌های قبلی گرفتند.



شکل ۱۱-۲: مقادیر اولیه ارزش‌ها در حالت تغییر داده شده

در سرعت همگرایی هیچ تأثیری دیده نشد و هر دو الگوریتم مانند دفعه قبل در ۵ مرتبه اجرا به همگرایی رسیدند.

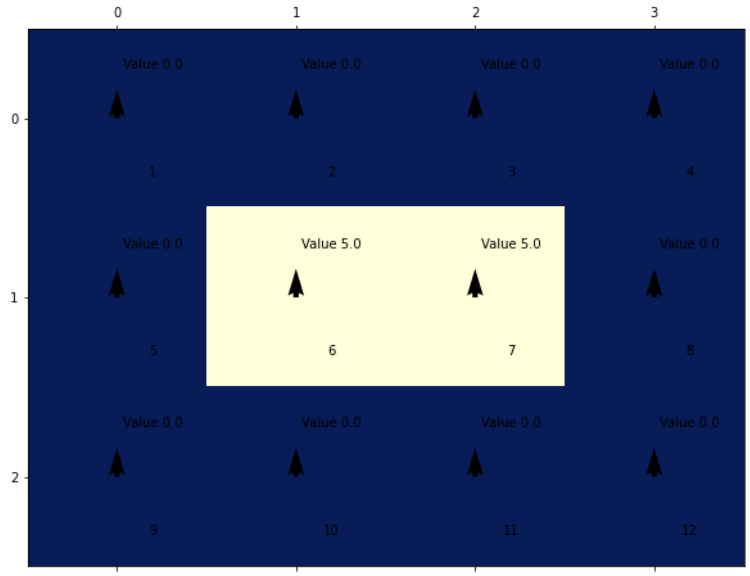
۲- کم کردن ارزش خانه‌های نزدیک به چاله: خانه‌های ۲و۷ که نزدیک به چاله هستند ارزش -۵ داده شد و مابقی خانه ارزش صفر گرفتند.



شکل ۱۲-۲: ارزش مقادیر در حالت دوم

در این حالت هم هیچ تفاوتی در نحوه همگرایی خانه مشاهده نشد و در نهایت سیاست نهایی نیز کاملاً مشابه حالت‌های قبلی بود.

۳- زیاد کردن ارزش خانه‌های نزدیک به خانه شروع که در مسیر خانه هدف هستند: خانه‌های ۶ و ۷ که در مسیر خانه شروع به خانه هدف بودند ارزش ۵ و مابقی خانه‌ها ارزش صفر گرفتند



شکل ۱۳-۲: مقادیر اولیه در حالت سوم

عمل‌کرد الگوریتم تفاوتی با دفعات قبل نداشت.

به طور کلی میتوان نتیجه گرفت که ارزش اولیه خانه در سرعت پیدا کردن سیاست بهینه و یا نرخ همگرایی هیچ تأثیری ندارد، دلیل این اتفاق این است که سیاست ما نقش اصلی در تعیین ارزش State ها دارد و حتی زمانی که ما ارزش حالت هارا متناسب با سیاست بهینه تغییر دهیم در Iteration اولی که الگوریتم اجرا شود ارزش حالت‌ها متناسب با سیاست تصادفی تغییر میکند و تغییر دادن ارزش‌های اولیه تأثیر به‌خصوصی در همگرایی الگوریتم نمی‌گذارد.

سوال ۳ -

هدف سوال:

در این سؤال به پیاده‌سازی یک مسأله MDP نسبتاً پیچیده‌تر در محیط پایتون می‌پردازیم و سعی میکنیم با الگوریتم های Dynamic Programming به حل آن‌ها بپردازیم و با تغییر پارامترهای محیط و الگوریتم تأثیر این پارامتر ها را در مدل مشاهده نماییم.

بخش اول-

برای پیاده‌سازی این مسأله کلاس FrozenLake به شکل زیر تغییر پیدا کرد:

تابع Init مقادیر شماره دانشجویی، گاما، تتا، احتمال سرخوردن، سایز نقشه و نحوه مقدار دهی احتمال شکستن خانه‌های ناامن (به شکل ثابت و برابر ۱ (حالت تصادفی) یا یک مقدارتصادفی (حالت ۲)) را به عنوان ورودی دریافت میکند.

سپس پارامترها مقدار دهی اولیه می‌شوند، برای مدل سازی این مسأله یک حالت اضافه بر خانه‌های موجود در نقشه گرفته می‌شود که به عنوان حالت Terminal از آن استفاده می‌شود، به طور مثال در نقشه ابتدایی که دارای ۳۶ خانه است یک خانه ۳۷ ام نیز برای مدل سازی حالت Terminal در نظر گرفته می‌شود و هنگامی که خانه ناامن شکسته می‌شود عامل به داخل خانه ۳۷ منتقل می‌شود. یک آرایه به نام Q تعریف میکنیم که ارزش هر اکشن را در هر حالت در آن ذخیره می‌کنیم. مقادیر پاداش و احتمال انتقال در دو آرایه به نام های R و T ذخیره می‌شوند ابعاد این دو آرایه به شکل

[state,Action,state] تعریف شده است به این صورت که state اولیه به عنوان حالت مبدأ و action عمل انجام شده و state دوم به عنوان حالت مقصد تعریف می‌شود، حال برای مقدار دهی عناصر این دو آرایه به این شکل عمل میکنیم که برای هر اکشن ابتدا چک میکنیم که عامل با انجام اکشن به فنس برخورد میکند یا خیر در صورتی که برخورد کند با احتمال ۱ به خانه اولیه بر می‌گردد (در آرایه T خانه مبدأ و مقصد یکی است و احتمال ۱ است) و هیچ پاداشی دریافت نمی‌کند اما در غیر این صورت با احتمال برابر با مقدار مشخص شده در خروجی تابع make_map به حالت Terminal که به صورت اضافی تعریف شده است می‌رود و با احتمال ۱ منهای این مقدار به خانه مقصد مورد نظر می‌رود، در صورتی که خانه مقصد شکسته شود عامل مقدار پاداش ۱۱- و غیر این صورت پاداش ۱- می‌گیرد.

همچنین برای خانه‌های هدف و Terminal به ازای تمامی اکشن ها با احتمال ۱ به خانه مبدأ باز میگردیم و هیچ پاداش دریافت نمیکنیم، برای خانه‌های بالا و سمت چپ خانه هدف نیز با انجام اکشن پایین و راست پاداش ۹۹ دریافت کرده و به خانه هدف می‌رویم.

```
for r in range(n_rows):
    for c in range(n_cols):
        """IF COLUMN INDEX OF THE CELL IS ZERO (aka cell is next to the left fence)
        FOR ACTION LEFT AGENT GETS BACK TO THE SAME CELL WITH PROB =1
        OTHERWISE AGENT WILL GO TO EITHER IT'S LEFT CELL OR TO TERMINAL STATE BASED ON THE LEFT CELL'S BREAKING PROBABILITY
        """
        if c==0:
            self.T[r*n_rows+c,self.ACTION['LEFT'],r*n_rows+c] = 1.0
        else:
            self.T[r*n_rows+c,self.ACTION['LEFT'],r*n_rows+c-1]= 1 - self.obs[r][c-1]
            self.T[r*n_rows+c,self.ACTION['LEFT'],self.states[-1]] = self.obs[r][c-1]
        #FOR CELLS NEXT TO RIGHT FENCE AGENT GOES BACK TO IT'S CELL WITH PROB 1
        if c==n_cols-1:
            self.T[r*n_rows+c,self.ACTION['RIGHT'],r*n_rows+c] = 1.0
        else:
            #OTHER WISE EITHER TO IT'S RIGHT CELL OR TO TERMINAL
            self.T[r*n_rows+c,self.ACTION['RIGHT'],r*n_rows+c+1]= 1 - self.obs[r][c+1]
            self.T[r*n_rows+c,self.ACTION['RIGHT'],self.states[-1]] = self.obs[r][c+1]
        #CELLS BELOW THE TOP FENCE
        if r==0:
            self.T[r*n_rows+c,self.ACTION['UP'],r*n_rows+c] = 1.0
        else:
            self.T[(r)*n_rows+c,self.ACTION['UP'],(r-1)*n_rows+c]= 1 - self.obs[r-1][c]
            self.T[r*n_rows+c,self.ACTION['UP'],self.states[-1]] = self.obs[r-1][c]
        #CELLS ABOVE THE BOTTOM FENCE
        if r==n_rows-1:
            self.T[r*n_rows+c,self.ACTION['DOWN'],r*n_rows+c] = 1.0
        else:
            self.T[(r)*n_rows+c,self.ACTION['DOWN'],(r+1)*n_rows+c]= 1 - self.obs[r+1][c]
            self.T[r*n_rows+c,self.ACTION['DOWN'],self.states[-1]] = self.obs[r+1][c]
#WHEN AGENT REACHES GOAL OR TERMINAL STATE IT CAN'T GET OUT WITH ANY ACTION SO WITH PROBABILITY OF 1 GOES BACK IT IT'S CELL
self.T[self.n_states-2,:,:] = 0
self.T[self.n_states-2,:,self.n_states-2] = 1
self.T[self.n_states-1,:,self.n_states-1] = 1
```

شکل ۱-۳: نحوه مقدار دهی آرایه T

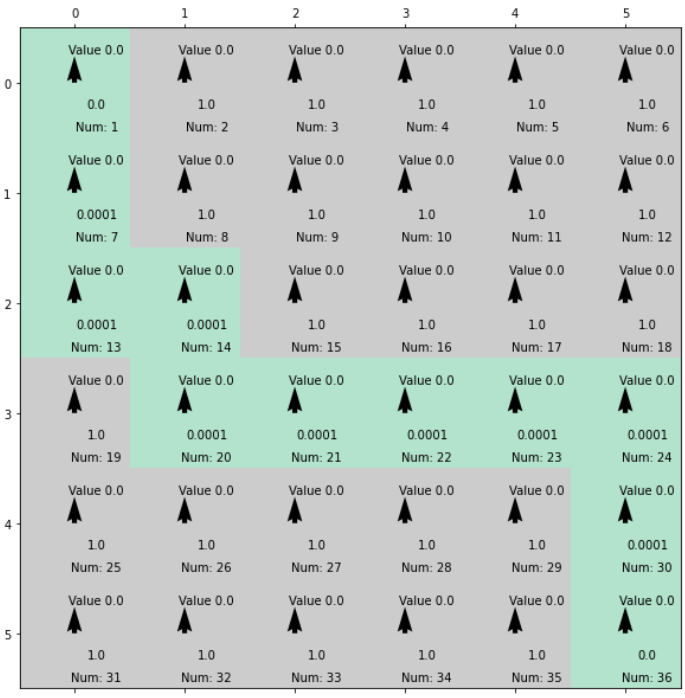
یک تابع reset تعریف میکنیم که در آن مقادیر پالیسی، ارزش و ارزش اکشن هارا صفر می‌کنیم.

تابع step نیز صرفاً جهت نمایش عمل‌کرد عامل با انجام هر اکشن نوشته شده است و با گرفتن اکشن مورد نظر حالت مقصد، پاداش و ترمینال بودن حالت را به ما بر می‌گرداند. جهت پیاده‌سازی شرایط سر خوردن عامل با توجه به احتمال سر خوردن عامل با احتمال مشخص شده اکشن مشخص شده را انجام داده و در غیر این صورت یکی از سه اکشن دیگر را به شکل تصادفی انجام می‌دهد.

```python
def step(self,input_action):
    #THIS FUNCTION IS JUST FOR DEMONSTRATION PURPOSES AND DOESN'T HAVE ANY ROLE IN SOLVING THE GIVEN PROBLEM
    assert input_action in self.ACTION, "Input Action must be LEFT,RIGHT,UP,DOWN"
    #GET THE ACTION
    action = self.ACTION[input_action]
    #SAVE THE LAST STATE
    old_state = self.s
    #WITH A GIVEN PROBABILITY SLIP OR DON'T SLIP
    if np.random.random() > self.slip_prob:
        #DON'T SLIP
        transition_prob = self.T[self.s,action,:]
        self.s = np.random.choice(self.states,p=transition_prob)
    else:
        #SLIP
        #CHOOSE A RANDOM ACTION
        action = np.random.choice(len(self.ACTION))
        #GET THE PROBABILITY
        transition_prob = self.T[self.s,action,:]
        #GO TO DESTINATION CELL  BASED ON THE TRANSITION PROBABILITY
        self.s = np.random.choice(self.states,p=transition_prob)
    #GET THE REWARD
    reward = self.R[old_state,action,self.s]
    #IF THE DESTINATION CELL IS TERMINAL (ICE BROKE) GO BACK TO INITIAL STATE
    terminate = False
    if self.s == self.states[-1]:
        terminate = True
        self.s = 0
    #RETURN THE STATE, REWARD AND IF IT TERMINATED
    return self.s,reward,terminate
```

شکل ۲-۳: پیاده‌سازی تابع Step

تابع render نیز سیاست نهایی و مقادیر بهینه ارزش هارا به شکل یک پلات نمایش می‌دهد. سیاست ابتدایی برای تمامی خانه‌ها اکشن بالا و مقادیر اولیه همه ارزش‌ها صفر است.



شکل ۳-۳: خروجی تابع Render برای محیط، مقادیر فلش جهت اکشن، مقدار Value ارزش هر حالت و عدد پایین سمت راست فلش احتمال شکستن و Num شماره خانه را نمایش می‌دهد مسیر امن با رنگ آبی مشخص شده است

الگوریتم Policy Iteration:

الگوریتم مورد نظر به شکل زیر پیاده‌سازی می‌شود:

```
tic = time()
iteration = 0
while True:
    iteration+=1
    #POLICY EVALUATION
    while True:
        delta=0
        #LOOP OVER ALL STATES
        for s in environment.states:
            #SAVE OLD VALUE FOR COMPUTING DELTA
            v_old = environment.V[s]
            #COMPUTE DISCOUNTED RETURN FOR EACH ACTION STATE
            G = (environment.discount_factor * environment.V + environment.R[s])
            #COMPUTE Q FOR EACH ACTION STATE
            environment.Q[s] = np.sum(environment.T[s] * G ,axis=1)
            #BASED ON THE SELECTED ACTION (ACTION WITH BIGGEST Q) SELECT OTHER 3 ACTIONS WITH EQUAL PROBABLITY
            slippery_action = np.full(4,fill_value=0.33)
            slippery_action[np.argmax(environment.policy[s])] = 0
            #CALCULATE AN EXPECTED VALUE BASED ON ACTUAL POLICY AND A RANDOM ACTION
            environment.V[s] = (1-environment.slip_prob) * np.sum(environment.Q[s]*environment.policy[s]) \
            + (environment.slip_prob) * np.sum(environment.Q[s] * slippery_action)
            #COMPUTE DELTA VALUE
            delta = max(delta,np.abs(v_old - environment.V[s]))
        #IF DELTA IS SMALLER THAN THETA END POLICY EVALUATION
        if delta < environment.theta:
            break
    #POLICY IMPROVEMENT
    policy_stable = True
    #LOOP OVER ALL STATES
    for s in environment.states:
        #SAVE THE OLD POLICY
        old_policy = np.copy(environment.policy[s])
        #CHANGE THE OLD POLICY TO THE ACTION WHITH BIGGEST Q VALUE
        environment.policy[s] = 0
        environment.policy[s][np.argmax(environment.Q[s])] = 1
        #IF THE POLICY IS CHANGED FOR THE STATE SET THE STABLE FLAG TO FALSE
        if np.allclose(old_policy,environment.policy[s])==False:
            policy_stable = False
    #IF POLICY IS STABLE END THE POLICY IMPROVEMENT
    if policy_stable:
        break
tac= time()
print(f'Policy Iteration Ended in {iteration} Iteration and {tac - tic} seconds')
```

شکل ۴-۳: پیاده‌سازی الگوریتم Policy Iteration

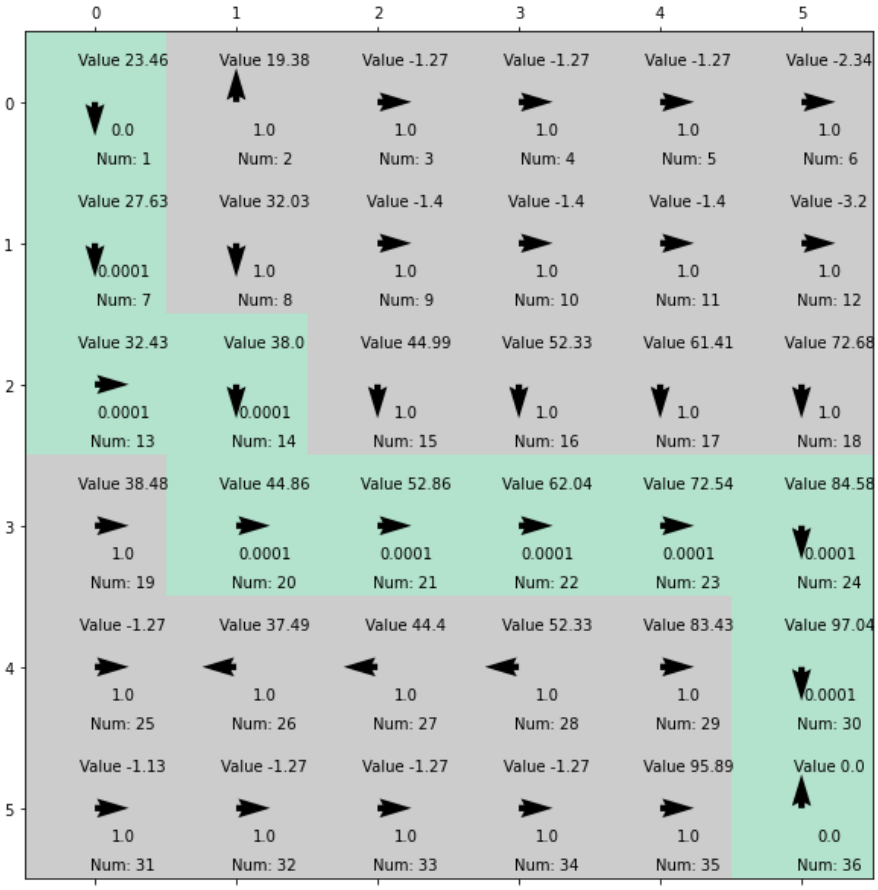
یک حلقه تا زمانی که سیاست پایدار شود (تغییراتی در دو مرتبه اجرای پشت سر هم نداشته باشد) ابتدا Policy evaluation و سپس Improvement را اجرا می‌کند، برای انجام Policy Evaluation بر روی تمامی حالت‌ها یک حلقه اجرا کرده و ابتدا مقدار Discounted Return را برای حالت مورد نظر حساب میکنیم سپس سپس مقدار Q را با ضرب ماتریس G در Transition probability محاسبه می‌کنیم حال برای در نظر گرفتن احتمال سرخوردن عامل مقدار ارزش حالت را یک بار برای زمانی که عامل اکشن مربوط به سیاست[s]policy و یک بار زمانی که عامل یکی از سه عمل دیگر را به شکل تصادفی

slippery_action [s] انجام دهد محاسبه میکنیم سپس متوسط وزن دار این دو مقدار را محاسبه میکنیم، سپس مقدار دلتا را محاسبه کرده و در صورتی که از مقدار تتا کمتر باشد از حلقه خارج می‌شویم.

حال برای تمامی حالت‌ها اکشن بهینه را بر اساس ماکزیمم مقدار Q به دست آمده برای هر اکشن محاسبه میکنیم و در صورتی که سیاست ما در تمامی حالت‌ها برابر Iteration قبلی بود الگوریتم را خاتمه می‌دهیم.

نتایج:

الگوریتم پس از ۷ مرتبه اجرا به سیاست بهینه رسید و متوقف شد، زمان اجرای الگوریتم۰.۲ ثانیه بود.



شکل ۵-۳: نتیجه ارزش‌ها و سیاست نهایی در الگوریتم Policy Iteration

از خروجی مشخص است که الگوریتم سیاست بهینه برای مسیر امن را به خوبی پیدا کرده است اما در خانه‌های ناامن آن‌هایی که نزدیک به یک خانه‌امن هستند سیاست بهینه رفتن به خانه امن است ولی خانه‌هایی که در همسایگی آن‌ها تنها خانه نا امن وجود دارد سیاست بهینه ندارند، دلیل این امر نیز این است که در صورتی حرکتی انجام دهند در هر صورت به حالت Terminal خواهند رفت.

	Left	Down	Right	Up
0	21.0841	23.8671	-1.0	21.0841
1	20.1102	-11.0	-1.0	17.4079
2	-11.0	-11.0	-1.0	-1.1389
3	-11.0	-11.0	-1.0	-1.1389
4	-11.0	-11.0	-1.0	-1.1389
5	-11.0	-11.0	-2.1052	-2.1052
6	24.8705	28.1814	-1.0	20.1102
7	23.8677	33.2	-1.0	-11.0
8	-11.0	-11.0	-1.0	-11.0
9	-11.0	-11.0	-1.0	-11.0
10	-11.0	-11.0	-1.0	-11.0
11	-11.0	-11.0	-2.8824	-11.0
12	29.1853	-11.0	33.201	23.8677
13	28.1814	39.366	-1.0	-11.0
14	33.2	46.5707	-1.0	-11.0
15	-11.0	54.8256	-1.0	-11.0
16	-11.0	64.2817	-1.0	-11.0
17	-11.0	75.1139	65.4142	-11.0
18	34.6286	-11.0	39.367	28.1814
19	-11.0	-11.0	46.5717	33.2
20	39.366	-11.0	54.8266	-11.0
21	46.5707	-11.0	64.2827	-11.0
22	54.8256	-11.0	75.1149	-11.0
23	64.2817	86.3255	76.1225	-11.0
24	-1.1389	-11.0	-1.0	-11.0
25	-11.0	-11.0	-1.0	39.366
26	-11.0	-11.0	-1.0	46.5707
27	-11.0	-11.0	-1.0	54.8256
28	-11.0	-11.0	86.3265	64.2817
29	-11.0	99.0	87.3352	75.1139
30	-1.0189	-1.0189	-1.0	-11.0
31	-11.0	-1.1389	-1.0	-11.0
32	-11.0	-1.1389	-1.0	-11.0
33	-11.0	-1.1389	-1.0	-11.0
34	-11.0	86.2999	99.0	-11.0
35	0.0	0.0	0.0	0.0
36	0.0	0.0	0.0	0.0

شکل ۶-۳: مقادیر State-Action ها (Q) برای تمامی حالت ها

همچنین مقادیر ارزش State-action در شکل ۶-۳ مشخص است، همانطور که از مقادیر پیداست مقادیری که در هر حالت عامل را به خانه امن بعدی می‌برد بیشترین ارزش را دارند.

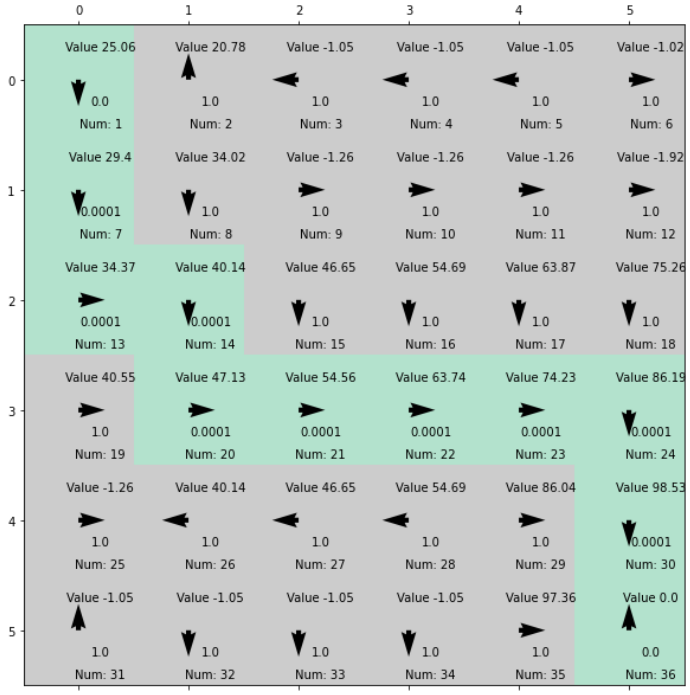
الگوریتم Value Iteration:

این الگوریتم به این شکل پیاده‌سازی شده است :

```python
#RESET THE VARIABLES
environment.reset()
tic = time()
iteration=0
while True:
    delta=0
    iteration+=1
    #LOOP OVER ALL STATES
    for s in environment.states:
        #SAVE THE OLD VALUE FOR CALCULATING DELTA
        v_old = environment.V[s]
        #CALCULATE THE DISCOUNTED RETURN
        G = (environment.discount_factor * environment.V + environment.R[s])
        #CALCULATE THE Q FOR ALL ACTIONS
        environment.Q[s] = np.sum(environment.T[s] * G ,axis=1)
        #BASED ON THE SELECTED ACTION (ACTION WITH BIGGEST Q) SELECT OTHER 3 ACTIONS WITH EQUAL PROBABLITY
        slippery_action = np.full(4,fill_value=0.33)
        slippery_action[np.argmax(environment.policy[s])] = 0
        #CALCULATE AN EXPECTED VALUE BASED ON ACTUAL VALUE AND VALUE IN CASE OF SLIPPING
        environment.V[s] = (1-environment.slip_prob) * np.max(environment.Q[s]) \
        + (environment.slip_prob) * np.sum(environment.Q[s] * slippery_action)
        #CALCULTATE DELTA VALUE
        delta = max(delta,np.abs(v_old - environment.V[s]))
    #IF DELTA IS SMALLER THAN THETA END THE WHILE LOOP
    if delta < environment.theta:
        break
#LOOP OVER ALL STATES FOR IMPROVEMENT
for s in environment.states:
    #CHANGE THE POLICY TO THE ACTION WITH BIGGEST Q
    environment.policy[s] = 0
    environment.policy[s][np.argmax(environment.Q[s])] = 1
tac = time()
print(f'Value Iteration Ended in {iteration} Iteration and {tac - tic} seconds')
```

شکل ۷-۳: پیاده‌سازی الگوریتم Value Iteration

در یک حلقه تا زمانی که مقادیر تغییر ارزش‌ها نسبت به اجرای قبلی از یک مقدار تتا بیشتر باشد به ازای تمامی حالت‌ها مقدار ارزش State-action هارا محاسبه میکند برای محاسبه ارزش هر state دو حالت را در نظر میگیریم ۱- حالتی که عامل سر نخورد که در این صورت ارزش برابر با ماکزیمم مقدار Q در آن حالت خواهد بود و احتمال ۹۶ درصد دارد ۲- حالتی که عامل سر بخورد که در این صورت با احتمال ۳۳ درصد یکی از سه اکشن دیگر را انتخاب خواهد کرد در این صورت میانگین ارزش State-Action را برای تمامی این اکشن ها محاسبه میکنیم و احتمال آن ۴ درصد خواهد بود، حال ارزش هر حالت را از متوسط وزن دار این دو حالت محاسبه میکنیم.



شکل ۸-۳: خروجی نهایی ارزش‌ها و سیاست بهینه در الگوریتم Value Iteration

همانطور که مشخص است خروجی سیاست بهینه توسط این الگوریتم کاملاً مشابه الگوریتم قبلی و برابر با سیاست بهینه است همچنین الگوریتم در ۱۳ مرتبه اجرا و ۰.۰۴ ثانیه اجرا به همگرایی رسید.

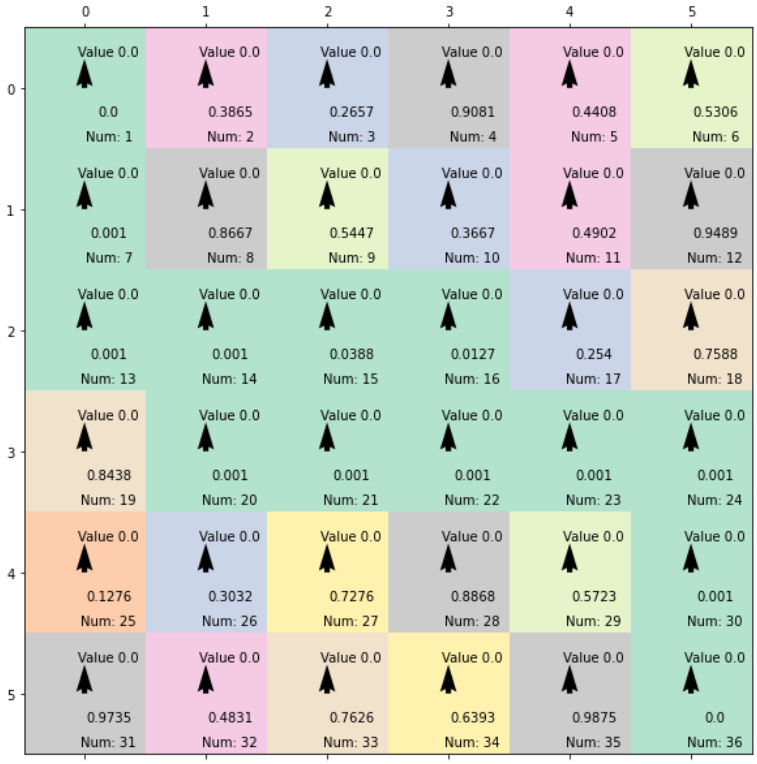
	Left	Down	Right	Up
0	22.5503	25.4583	-1.0	22.5503
1	21.5544	-11.0	-1.0	18.6978
2	-11.0	-11.0	-1.0	-0.9144
3	-11.0	-11.0	-1.0	-0.9144
4	-11.0	-11.0	-1.0	-0.9144
5	-11.0	-11.0	-0.8853	-0.8853
6	26.4619	29.9332	-1.0	21.5544
7	25.4584	35.1242	-1.0	-11.0
8	-11.0	-11.0	-1.0	-11.0
9	-11.0	-11.0	-1.0	-11.0
10	-11.0	-11.0	-1.0	-11.0
11	-11.0	-11.0	-1.6764	-11.0
12	30.9373	-11.0	35.1252	25.4584
13	29.9332	41.4114	-1.0	-11.0
14	35.1242	48.0992	-1.0	-11.0
15	-11.0	56.3612	-1.0	-11.0
16	-11.0	65.7949	-1.0	-11.0
17	-11.0	76.5665	67.7371	-11.0
18	36.4972	-11.0	41.4124	29.9332
19	-11.0	-11.0	48.1002	35.1242
20	41.4114	-11.0	56.3622	-11.0
21	48.0992	-11.0	65.7959	-11.0
22	56.3612	-11.0	76.5675	-11.0
23	65.7949	87.6653	77.5752	-11.0
24	-1.1372	-11.0	-1.0	-11.0
25	-11.0	-11.0	-1.0	41.4114
26	-11.0	-11.0	-1.0	48.0992
27	-11.0	-11.0	-1.0	56.3612
28	-11.0	-11.0	87.6663	65.7949
29	-11.0	99.0	88.6752	76.5665
30	-0.9144	-0.9144	-1.0	-11.0
31	-11.0	-0.9144	-1.0	-11.0
32	-11.0	-0.9144	-1.0	-11.0
33	-11.0	-0.9144	-1.0	-11.0
34	-11.0	87.6224	99.0	-11.0
35	0.0	0.0	0.0	0.0
36	0.0	0.0	0.0	0.0

شکل ۹-۳: مقادیر نهایی ارزش State-Action در هر حالت

با بررسی مقادیر ارزش State-action ها مشخص است که ارزش هر حالت متوسط ماکسیمم ردیف و میانگین ارزش ۳ اکشن دیگر در آن حالت مذکور می‌باشد.

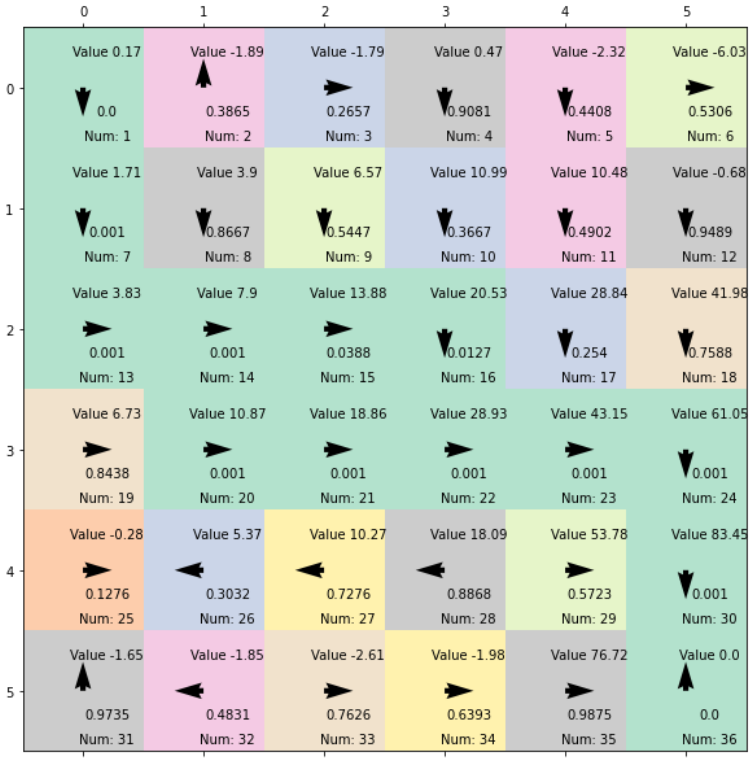
بخش دوم:

با تغییر تابع make_map و اضافه کردن قابلیت تغییر سایز و همچنین امکـان تغیـیر حـالت مقـدار دهی احتمال شکستن نقشه جدید طبق صورت سؤال ساخته شده است.



شکل ۱۰-۳: نقشه جدید با مقادیر اولیه

الگوریتم Policy Iteration برروی این نقشه اجرا شد،با ۶ مرتبه اجرا در ۰.۱۴ ثانیه الگوریتم به همگرایی رسید.



شکل ۱۱-۳: خروجی نهایی الگوریتم Policy Iteration

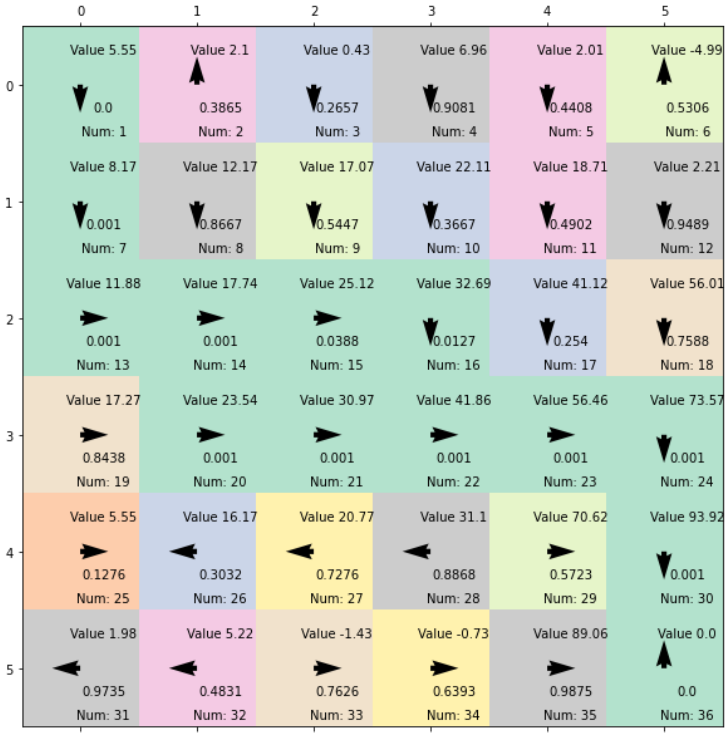
همانطور که مشخص است خانه‌هایی که در مسیر امن هستند (به جز خانه ۱۴) همگی سیاست بهینه را پیدا کردند اما خانه‌های دور تر از مسیر امن اکثراً اکشن اشتباه را انتخاب نموده اند، دلیل کاهش دقت الگوریتم در این نقشه میتواند ۱- احتمال بالای سرخوردن ۲- کم شدن احتمال شکسته شدن یخ باشد که باعث می‌شود که مقادیر Q در هر حالت خیلی نزدیکتر به هم شده و گاهی اوقات حتی برای اکشن های غیر بهینه بزرگ‌تر شود.

	Left	Down	Right	Up
0	0.0826	0.5176	-2.0854	0.0826
1	-0.845	-9.1994	-2.1867	-1.7691
2	-5.9106	-3.7569	-0.9614	-1.616
3	-4.8397	1.5981	-2.1663	0.4194
4	-10.0419	-1.0937	-3.5456	-2.0855
5	-6.574	-10.5202	-5.4233	-5.4233
6	1.5291	2.427	-0.5327	-0.845
7	0.531	6.0942	1.69	-5.9106
8	-9.1986	10.6196	5.2649	-4.8397
9	-3.7566	17.1148	3.8087	-10.0419
10	1.5981	15.823	-1.0312	-6.574
11	-1.0937	0.5231	-0.611	-8.8516
12	3.4405	-8.4913	6.1042	0.531
13	2.4295	8.758	11.0073	-9.1986
14	6.0947	15.9457	17.2422	-3.7566
15	10.6197	25.0046	18.3626	1.5981
16	17.1148	37.7837	8.1115	-1.0937
17	15.823	53.878	37.782	-10.5202
18	6.0565	-2.5019	8.768	2.4295
19	-8.4912	-0.6653	15.9557	6.0947
20	8.7595	-5.7582	25.0146	10.6197
21	15.9459	-8.0251	37.7937	17.1148
22	25.0046	13.9767	53.888	15.823
23	37.7837	74.0229	54.943	0.5231
24	-0.2589	-10.7759	2.3663	-8.4912
25	-2.4933	-6.6949	1.5178	8.7595
26	-0.6624	-9.1851	0.8428	15.9459
27	-5.7581	-8.034	19.6999	25.0046
28	-8.0251	-10.0121	74.0329	37.7837
29	13.9767	99.0	75.108	53.878
30	-1.5275	-1.5275	-1.8643	-2.4933
31	-10.7748	-1.6719	-1.5587	-0.6624
32	-6.6914	-2.3539	-1.6415	-5.7581
33	-9.1849	-1.7782	-0.137	-8.0251
34	-8.034	69.052	99.0	13.9767
35	0.0	0.0	0.0	0.0
36	0.0	0.0	0.0	0.0

شکل ۱۲-۳: مقادیر ارزش State-action ها در هر حالت

الگوریتم Value Iteration:

الگوریتم برروی نقشه جدید اعمال شد و در ۱۵ مرتبه اجرا و ۰.۰۴ ثانیه به همگرایی رسید.



شکل ۱۳-۳: خروجی سیاست و ارزش نهایی حالت ها

خروجی نهایی سیاست مانند مثال‌های قبلی کاملاً شبیه به الگوریتم Policy Iteration می‌باشد و سیاست نهایی مانند قسمت قبل برای حدوداً تمامی خانه‌ها بهینه است اما در خانه‌های دور تر از مسیر امن سیاست بهینه حاصل نشده است.

	Left	Down	Right	Up
0	4.9431	6.3221	0.1282	4.9431
1	3.9981	-8.2073	-0.714	1.8389
2	-3.7068	0.547	-0.4237	0.3895
3	-3.3699	7.9379	0.0138	6.2677
4	-9.5042	2.6813	-3.1092	1.8129
5	-4.3941	-10.3871	-4.4936	-4.4936
6	7.3395	9.6703	0.4594	3.9981
7	6.3337	14.944	5.9939	-3.7068
8	-8.2068	20.3412	11.6047	-3.3699
9	0.5472	27.9152	7.5837	-9.5042
10	7.9379	24.0719	-0.8981	-4.3941
11	2.6813	3.5673	1.9931	-8.4158
12	10.691	-7.0089	14.954	6.3337
13	9.6726	20.1502	20.7289	-8.2068
14	14.9444	26.8371	28.0426	0.5472
15	20.3413	36.6278	26.6115	7.9379
16	27.9152	49.7538	11.1557	2.6813
17	24.0719	65.1367	50.4054	-10.3871
18	15.5438	2.0819	20.1602	9.6726
19	-7.0088	6.1083	26.8471	14.9444
20	20.1504	-3.1832	36.6378	20.3413
21	26.8372	-6.6992	49.7638	27.9152
22	36.6278	20.4586	65.1467	24.0719
23	49.7538	83.4313	66.2129	3.5673
24	4.9954	-10.6883	9.14	-7.0088
25	2.0822	-3.4039	4.0928	20.1504
26	6.1084	-8.9318	2.1688	26.8372
27	-3.1832	-7.6284	26.1818	36.6278
28	-6.6992	-9.8733	83.4413	49.7538
29	20.4586	99.0	84.5258	65.1367
30	1.7829	1.7829	1.4267	2.0822
31	-10.6883	4.6944	-1.3055	6.1084
32	-3.4039	-1.287	-1.2359	-3.1832
33	-8.9318	-0.6539	0.0017	-6.6992
34	-7.6284	80.1556	99.0	20.4586
35	0.0	0.0	0.0	0.0
36	0.0	0.0	0.0	0.0

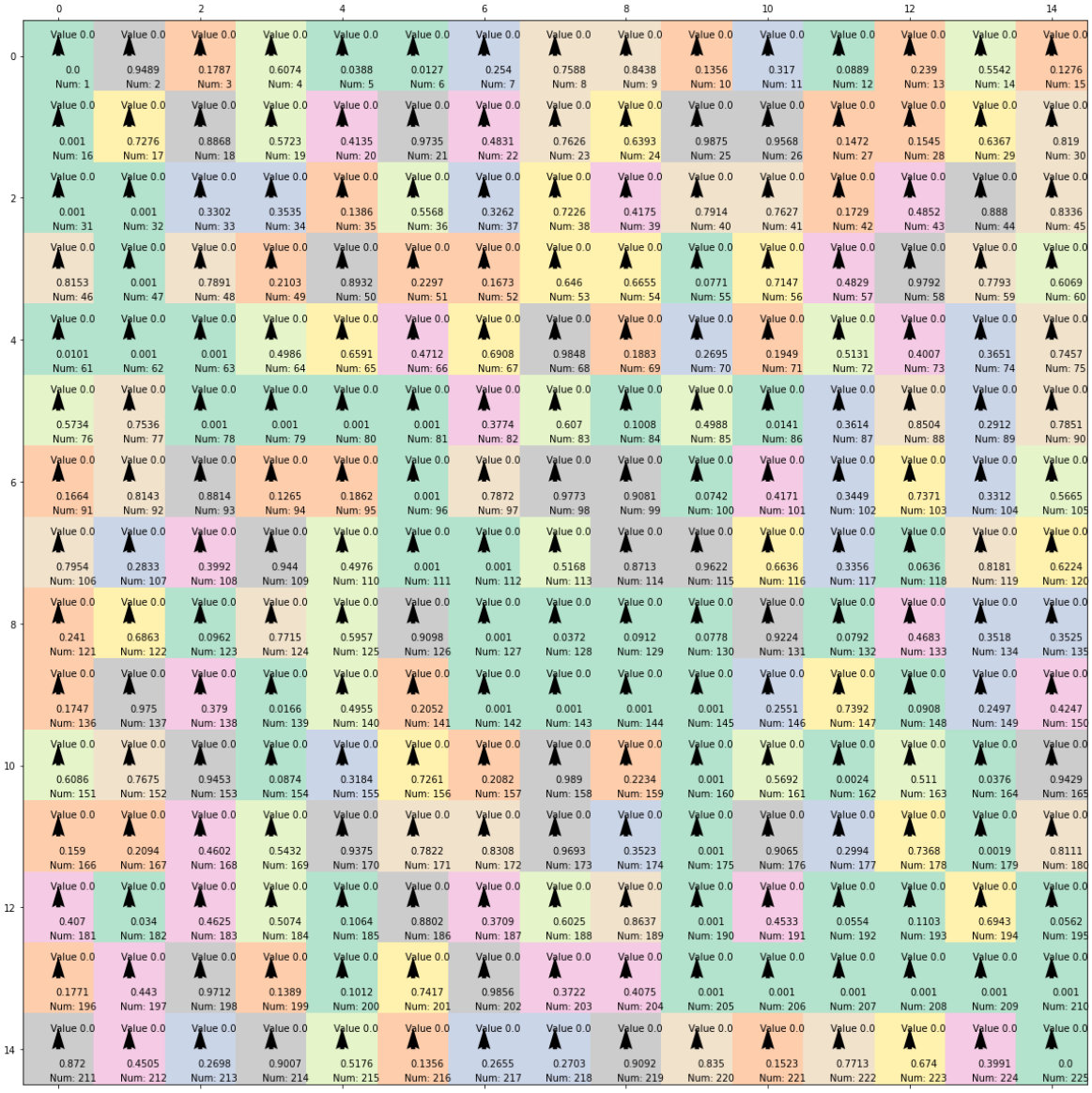
شکل ۱۴-۳: ارزش State-action ها در هر حالت در الگوریتم Value Iteration

سرعت همگرایی در قسمت قبلی برای الگوریتم Policy Iteration در ۷ مرتبه اجرا و ۰.۱۸ ثانیه بود ولی در این قسمت ۶ مرتبه و ۰.۱۴، همچنین در الگوریتم Value Iteration در قسمت قبلی با ۱۳ مرتبه اجرا و ۰.۰۴ ثانیه و در این قسمت با ۱۵ مرتبه اجرا و ۰.۰۴ ثانیه به همگرایی رسید.

که میتوان نتیجه‌گیری کرد که در حالت کلی الگوریتم Value Iteration از Policy Iteration سریع‌تر است اما در نقشه قبلی هر دو الگوریتم کمی دیرتر به همگرایی می‌رسند.

بخش سوم:

نقشه اولیه به شکل زیر است:



شکل ۱۵-۳: نقشه بخش سوم به همراه مقادیر اولیه سیاست و ارزش ها

الف) هردو الگوریتم Policy Iteration و Value Iteration با مقادیر زیر اجرا شدند:

۱, ۰.۹ ,۰.۵ 1e-1, 1e-2,1-e3,1e-4,1e-5,1e-6

(به دلیل اندازه بزرگ تصویر ها، تصویر سیاست خروجی برای هر الگوریتم در فولدر مخصوص خود به شکل جداگانه همراه گزارش آورده شده است)

سیاست نهایی خروجی توسط هیچ کدام از دو الگوریتم سیاست بهینه نیست و همچنین تغییر در مقدار تتا نیز تغییری در سیاست نهایی در الگوریتم ها به‌وجود نمی‌آورد و خروجی هر الگوریتم به ازای مقادیر مختلف تتا کاملاً مشابه است، مقدار پارامتر تتا در هر الگوریتم بایستی به گونه ایی انتخاب شود که اکشن

بهینه نسبت به مابقی اکشن ها ارزش بالاتری داشته باشد در صورتی که مقادیر ارزش State-action خیلی نزدیک به همدیگر باشند کاهش مقدار تتا ممکن است باعث شود سیاست بهینه انتخاب گردد اما در بقیه موارد تأثیر زیادی در انتخاب اکشن بهینه نخواهد داشت.

شکل ۱۶-۳: خروجی الگوریتم Policy Iteration به ازای تتا برابر با 1 e-6

شکل ۱۷-۳: خروجی الگوریتم Value Iteration به ازای تتا برابر با 1 e-6

ب) خیر همانطور که توضیح داده شد هیچ کدام از الگوریتم ها با مقادیر مختلف به سیاست بهینه دست پیدا نکردند، در این قسمت با تغییر پاداش خانه هدف از ۱۰۰ به ۳۰۰۰۰ میتوان به سیاست بهینه رسید، با تغییر پاداش به این مقدار هر دو الگوریتم با مقدار تتا برابر با ۱ به سیاست بهینه می‌رسند، با توجه به این موضوع می‌توان نتیجه گرفت که به دلیل مسافت زیادی که از خانه هدف تا خانه شروع وجود دارد مقدار ارزش State-action در خانه‌های اولیه پاداش خانه هدف را تقریباً در نظر نمی‌گیرند و برای محاسبه ارزش از ارزش خانه‌های نزدیک‌تر به خود استفاده می‌کنند که باعث می‌شود به سیاست بهینه دست پیدا نکنند.

شکل ۱۸-۳: خروجی الگوریتم Policy iteration به ازای پاداش ۳۰۰۰۰ و تتا برابر با 1e-6

شکل ۱۹-۳: خروجی الگوریتم Value Iteration به ازای پاداش ۳۰۰۰۰ و تتا برابر با 1e-6

نکات مهم و موارد تحویلی

لازم است که به نکات زیر در نوشتن گزارش توجه داشته باشید.

۱. ساختار کلی گزارش که در این فایل به آن اشاره شده باید رعایت شود. در صورت تمایل می‌توانید از latex یا هر نرم افزار دلخواه دیگر برای نوشتن گزارش استفاده کنید، به شرط اینکه ساختار کلی گفته شده رعایت شود. لذا در صورت رعایت نکردن ساختار کلی گزارش بخشی از نمره تمرین کم خواهد شد.
۲. برای تصاویر و جداول موجود در گزارش حتما کپشن قرار داده شود.
۳. **نتایج و تحلیل‌های** شما در روند نمره دهی اهمیت بسیار بالایی دارد، لذا خواهشمندیم کلیه نتایج و تحلیل‌های خواسته شده به صورت کامل و دقیق در گزارش آورده شوند.
۴. در صورت مشاهده شباهت بین گزارش شما و افراد مختلف نمره این سری تمرین برای شما در نظر گرفته نمی‌شود.

موارد تحویلی

۱. برای هر سری از تمرینات، فقط یک فایل با فرمت PDF آماده کنید.
۲. به همراه فایل گزارش، یک پوشه به نام Codes ایجاد کنید و کدها و فایل‌های پیاده‌سازی هر سوال را به صورت تفکیک شده در پوشه‌های جداگانه قرار دهید.
۳. هیچ گونه جدول یا تصویر به صورت جداگانه خارج از گزارش ارسال نشود. مگر اینکه به صورت صریح در تمرین از شما خواسته شده باشد.
۴. در انتها، لطفا برای هر تمرین گزارش و پوشه کدها را به صورت گفته شده، در یک فایل زیپ با فرمت زیر در سامانه یادگیری الکترونیک بارگذاری نمایید.

HW#_LastName_StudentNumber.zip

به طور مثال:

HW1_Mesbah_810111111.zip

منابع

در این بخش منابع (مقالات، سایت‌ها و ...) که در تمرینات و پیاده‌سازی استفاده کرده‌اید را به یک فرمت استاندارد برای مثال فرمت IEEE وارد کنید. برای رفرنس دهی میتوانید از بخش Reference نرم‌افزار Word یا این سایت یا افزونه‌های دیگر استفاده کنید. لازم است منبع مورد استفاده خود را در بخش مربوطه ارجاع دهید.

توجه به این نکته ضرروی است که میزان شباهت کد یا راه حل شما در صورت استفاده از منابع دیگر باید به حد معقولی باشد و کپی کردن از منابع مد نظر ما نیست. در صورت مشاهده کپی از منابع به صورت کامل نمره تمرین یا بخش مورد نظر به شما تعلق نمی‌گیرد.