: Simulated Annealing
این مکانیزم به طور میانگین چه تاثیری روی الگوریتم میگذارد؟ وجود چنین مکانیزمی در این بازی نیاز است؟
این مکانیزم باعث می شود که حرکات رندوم بیشتر بشود و قبل از برخورد به موانع،حرکت های رندوم مسیر دیگری را انتخاب می کنند.این مکانیزم می تواند کمک خوبی بکند اما در عین حال می تواند حرکت های رندوم و به درد نخور زیادی را بزند.

به نظر شما این

استراتژی در این الگوریتم استفاده خواهد شد؟
آیا
ممکن است در این الگوریتم در لوپ گیر کنیم؟

با توجه به کدی که زده شده، تنها زمانی که دما ۰ شده است ممکن است در لوپ گیر کنیم.
این استراتژی هم نباید احرا شود زیرا که باعث می شود حرکت رندوم زیاد شود و لوکال سرچ ما را خراب کند. اگر با یک احتمالی حرکت بد قبول نشد، باید ثابت بماند و منتظر حرکت بعدی بشود.

الگوریتم ژنتیک:

به نظر شما پیاده سازی چنین مکانیزمی نیاز است؟
این مکانیزم به طور میانگین باعث بهبود الگوریتم میشود؟

از آنجایی که الگوریتم بنده تمام مسیر را بر میگرداند، اصلا نیازی به این مکانیزم ها نیست.

سوال بعدی هم همین جواب بالایی را دارد.

پس از پیاده سازی نتایج را باهم بررسی کرده و به سوالات زیر پاسخ دهید:

- در کدام یک از این الگوریتم‌ها احتمال گیر افتادن در بهینه محلی بیشتر است؟
- هر الگوریتم چگونه با این مشکل مقابله می‌کند؟ آیا تپه نوردی با مکانیزم تصادفی به درستی از گیر افتادن در لوپ جلوگیری می‌کند؟
- کدام الگوریتم‌ها در یک محیط پیچیده و پر از موانع عملکرد بهتری دارند؟
- کدام یک از الگوریتم‌ها سریعتر به یک نتیجه نهایی می‌رسد؟
- کدام الگوریتم شانس بیشتری برای پیدا کردن بهینه جهانی دارد؟

نام مدرس: دکتر محمدی **دستیاران آموزشی مرتبط: زینب باقیان، کسری شریعتی** **مهلت تحویل:** 19 آبان	**تمرین عملی سری دوم** **درس هوش مصنوعی**	

- در هر الگوریتم، چگونه تنظیمات پارامترها (مانند نرخ جهش در الگوریتم ژنتیک یا دما در Simulated Annealing بر عملکرد آن تأثیر می‌گذارد؟ میتوانید این مورد را به صورت عملی پیاده سازی کرده و نتایج را باهم مقایسه کنید.

در نهایت تعداد گام‌های لازم برای رسیدن به گروگان، زمان اجرای الگوریتم و تعداد برخورد با موانع برای 3 الگوریتم را مقایسه کنید.(در صورت نیاز میتوانید پارامتر های مربوط به ساخت بازی را نیز تغییر دهید.)

۱- هیل کلایمبینگ زیرا به طور منطقی و تقریبا مصمم (غیر رندوم) مسیر خود را می سازد و در بهینه محلی گیر می افتد.
۲-تپه نوردی با حرکت رندوم تلاش می کند حل کند این مشکل را. منطقا اگر درست ایمپلمنت شده باشد، از لوپ خارج خواهد شد.
سیمولیتد انیلینگ با انتخاب رندوم و احتمال رفتن راه غلط از بهینه محلی فرار می کند. ژنتیک از میوتیشن و کراس اور استفاده می کند.
۳-با توجه به پیاده سازی بنده، احتمالا سیمولیتد انیلینگ بهتر کار میکند. ژنتیک من بهتر کار نمیکند زیرا من اجازه داده ام تا نسل اولیه ام از موانع رد شود و اگر موانع زیاد باشد، نیاز زیادی به میوتیشن می باشد که به نسبت زیاد بهینه نیست. هیل کلایمینگ هم که تقریبا رو هوا خواهد بود. گیر خواهد کرد و حرکات کامل رندوم میزند.
۴- هیل کلایمبینگ یا ژنتیک. هر کدام در یک سناریویی سریعتر به نتیجه نهایی می رسند. در کل ولی فکر کنم هیل کلایمینگ سریع تر می رسد.
۴-سیمولیتد و ژنتیک. بستگی به ایمپلنت دارن واقعا. اگر نسل اول ژنتیک خیلی ناب باشد خب بهینه جهانی را پیدا می کند. اگر نه، سیمولیتد به دلیل رندوم های زیادش بهینه جهانی را پیدا میکند (تقریبا)
۵-پایین بردن سرعت کاهش دما در سیمولیتد باعث می شود که تقریبا از وسط راه تبدیل به هیل کلایمینگ بشود. در یک سری موارد خوب و دیگر موارد بد است. به طور کلی و نظر شخص بنده، این است که قبل از ۰ شدن دما ما باید به جواب برسیم. اگر در وسط راه ۰ بشود و روش هیل کلامینگ بشود، ممکن است در لوپ هایی بد گیر بکند و بیرون آمدن از آن ها دشوار باشد. برای ژنتیک، زیاد کردن میوتیشن را دوست داشتم. باعث می شد بهتر برنامه کار بکند. اما به دلیل یکم باگی بودن کد بنده که در ادامه توضیح داده می شود، این مورد باعث می شود که برنامه به نسبت زیادی کند بشود.

نتایج نهایی:

هیل کلایمینگ:

این روش ساده بود و نکته خاصی نداشت. تنها نکته خاصی که داشت، نوع تشخیص لوپ بود. به دلیل طبیعت راه حل بهینه، دو بار در یک خانه رفتن را لوپ در نظر گرفتن کافی بود.

سیمولیتد:

این روش هم پیچیدگی خاصی نداشت. نکاتی که در کد رعایت شده این است: اگر با مسیر بد انتخاب شد و احتمال جواب نداد، حرکت رندوم نمی کند و سر جای خود می ایستد. برای اینکه به اشتباه لوپ تشخیص ندهد و حرکت رندوم بزند، کد را تغییر داده ام ذره ای. یک تابع هم زده ام که مسیر نهایی و واقعی را نشان می دهد(بعد از تمام شدن نشان دادن تمام حالات اتومات بالا می آید). مسیر نهایی خیلی بهینه است و به شخصه دوست داشتم این روش را.

ژنتیک:

۱۰۰ برابر دو روش بالا بیشتر چالش برانگیز بود. نسل اول از موانع رد می شود. برای حل مشکل موانع، از تابع میوتیشن به خصوص استفاده می شود. این تابع یکی از موانعی که ژن به آن برخورد کرده است را بر میدارد و سعی میکند که آن را دور بزند. به دلیل کمال گرایی غیر منطقی ام، خیلی کد را پیچیده زدم و تلاش کردم که میوتیشن را بسیار خوب بزنم. ولی یکی از باگ هایی که کد داشت و به زور درست کردم، این بود که بعضی وقت ها نمی توانست مانع را دور بزند و و به مسیر قبلی ملحق بشود و دچار اورفلو می شد. این تابع روش خیلی ساده تری می شود کدش را زد و هینت هایی از آن روش ساده در کد آلردی هست. اما به دلیل زمان کم این تمرین، فرصت نشد هم روش خودم رو کامل دیباگ کنم هم این روش ساده رو با روش خودم ترکیب کنم و یک میوتیشن ناب تحویل بدهم.

قسمت کراس اور مشکل خاصی نداشت.

برای ران شدن بهتر، جای حداقل ۱۰۰ بار ران شدن، بهتر بود ترش هلدی گذاشته می شد که اگر فیتنس بهترین ژن در ۱۰ ایتریشن وایل تغییر نکرده است، دیگر جواب را پیدا کرده ایم. اگر ۱۰ دقیقه بیشتر وقت بود این را هم زده بودم.

در کل الگوریتم خوبی از آب در آمده است. یک باگ کوچولویی که از میوتیشن ناقص من شاید به آن بر بخورد این است که به دلیل اینکه شاید به نتواند دور بزند، هیچوقت بهترین مسیر ساخته نمی شود و جواب نهایی از مانع می گذرد. این حالت کم پیش می آید ولی اگر پیش آمد تقصیر زمان کم این تمرین و فشردگی تمرین ها و امتحان های واحد های دیگر بگذارید!

زمان اجرای ژنتیک به نسبت فکر کنم سریع تر باشد (اگر میوتیشن و ترش هلد کامل درست شده باشند)