به نام خدا

امیرحسین عباسکوهی 810197539 گزارش کار پروژه 1 هوش مصنوعی

مقدمه:

در این تمرین قرار است به بررسی روش های مختلف جستجو برای حل مسئله، نحوه پیاده سازی ها و تمایز و برتری هر یک نسبت به دیگری را مورد بررسی قرار برهیم.

مدل کردن مسئله:

هر state در مسئله با توجه به مکان دانه ها و مشخصات بدن مار مشخص میشوند. البته در برخی شرایط که ادامه ذکر خواهند شد، اطلاعات بیشتری به مسئله اضافه می کنیم. در تمام روش ها اما ویژگی مختصات اجزای مار و مختصات و امتیاز دانه ها وجود دارد.
برای هر مار اطلاعات مهمی نیاز داریم: مکان سر مار و مکان دم مار که برای بررسی برخورد ها اهمیت دارد، یک لیست برای مشخصات بدنه مار که برای مشخص کردن استیت هست و البته برای بررسی برخورد ها اهمیت بالایی دارد. همچنین طول مار نیز ذخیره میشود.
برای دانه ها از یک dictionary استفاده میکنیم که در آن کلید مختصات دانه و مقدار امتیاز دانه می باشد.
حال برای هر استیت یک مار، یک دانه، عمق و البته اینکه استیت اولیه است یا خیر مورد بررسی قرار میگیرد. استیت اولیه برای حل مشکل نخوردن دانه در همان استیت اول قرار داده شده است.
در تمامی روش ها به جز IDS استیت یکسان استیتی با دانه های یکسان و مار یکسان می باشد، اما در IDS علاوه بر اینها عمق هم اهمیت دارد زیرا دیدن یک استیت یکسان در عمق مثل n و دیدن در عمق k دارد(یعنی اگر حداکثر عمق مثلا 10 باشد و یک استیت را در لول 9 دیده باشیمو جواب نرسیده باشد ممکن است دیدن همان استیت در لول 8 موجب رسیدن ما جواب شود و این باید در نظر گرفته شود).
با توجه به اینکه حرکت در بازی متوجه مار میباشد، پس برای کلاس Snake یکسری متد برای حرکت و خوردن دانه و ... استفاده میشود. یک متد به نام didGetSeed وجود دارد. در این متد بررسی میشود که آیا سر مار بر روی دانه قرار دارد یا خیر. در صورت وجود داشتن True و در غیر اینصورت False بازگردانده میشود. متد calNewSnake بر اساس اینکه مار دانه خورده است یا خیر و نحوه حرکت مختصات مار جدید(شامل اطلاعات سر ، دم و بدن) را باز میگرداند. این متد خود از متد move استفاده می کند. متد move بی توجه به برخورد به دانه مار را در جهت حرکت جا به جا میکند به این صورت که دم حذف شده و مختصات جدید سر در ابتدای body قرار داده میشود. در صورت برخورد به دانه خود calNewSnake مختصات دم حذف شده را به لیست باز می گرداند.برای هر مار یک متد isValidMove هم وجود دارد که بررسی میکند که آیا این حرکت مجاز هست فرضا اینکه آیا به بدن خود برخورد میکند یا خیر.
هر استیت جدید توسط متد calNewState در کلاس State محاسبه میشود. به این ترتیب که ابتدا مشخص میشود که آیا به دانه برخورد کردیم در صورت برخورد از امتیاز دانه مربوطه یک واحد کم میشود(اگر امتیاز صفر شود دانه حذف خواهد شد) و مختصات مار جدید با اضافه شدن به طول توسط calNewSnake مشخص میشود. در غیر اینصورت فقط calNewSnake بدون افزودن به طول کار خود را انجام میدهد.

در کلاس State یک متد isGoal وجود دارد. این متد اگر فقط یک دانه با امتیاز یک در زمین باشد و دقیقا سر مار روی آن باشد True باز میگرداند.
برای intial State هم در ابتدا dictionary و شی Snake بر اساس اطلاعات ورودی ساخته میشود و البته isInitial برای آن برابر یک قرار میگیرد.

توضیح الگوریتم های پیاده سازی شده:

- **الگوریتم bfs:** با توجه به شبه کد این الگوریتم، به این صورت عمل میکنیم که در ابتدا initial state در صف frontier قرار میگیرد. تا زمانی که صف خالی نشده باشد در هر مرحله state ابتدای صف خارج میشود وبه ازای تمام حرکات(چپ، راست، بالا و پایین)بررسی میشود که آیا این حرکت مجاز هست یا نه. در صورت مجاز بودن استیت جدید محاسبه میشود. اگر این استیت تکراری نبود به صف اضافه شده و البته به explored set هم اضافه میشود.
توجه کنید که در الگوریتم گفته شده است که استیت تکراری زمانی مشخص میشود که آن استیت نه در explored set باشد و نه در صف، اما با توجه به اینکه صف برای جستجو کند است باید چاره ای اندیشیده شود. برای حل این موضوع به جای آنکه در explored set هر استیتی که خارج میشود را قرار دهیم، هر استیتی که ساخته میشود(در حلقه حرکات) را قرار میدهیم به این صورت سرعت بهتری خواهیم داشت.

- **الگوریتم ids:** در این الگوریتم ما یک تابع داریم که به صورت حلقه بی نهایت تا زمانی که به جواب برسد به تابع depth limited search را صدا میکند. در هر بار صدا کردن explored set خالی میشود.در این تابع depth limited search در هر مرحله نود مربوط به explored اضافه شده یک حرکت انجام شده و تابع روی استیت جدید(البته باز هم اگر تکراری نباشد و حرکت مجاز باشد) فراخوانی می شود و البته عمق باقی مانده یک واحد کم میشود. اگر به جواب برسیم جواب چاپ شده و از حلقه خارج میشویم. اگر عمق به پایان برسد limit reach رسیدیم و این مقدار بازگردانده میشود. اگر تمامی بچه های یک نود به limit برسند و جواب نباشد fail بازگردانده میشود.

- **الگوریتم A*:** همانند الگوریتم bfs اینجا هم همان وضعیت را داریم. در اینجا فقط استیت های خارج شده از صف را در explored میریزیم. از طرفی در اینجا صف عادی نداریم و priority queue داریم. مقدار priority برای هر یک همان f است که برابر g که همان عمق استیت(State.depth) است، به اضافه h که همان heuristic میباشد. در اینجا از دو heuristic استفاده کرده ایم:

1- در این تابع heuristic ما به این صورت عمل میکنیم که فاصله همیلتونی سر مار از نزدیک ترین دانه را حساب میکنیم. سپس فاصله همیلتونی آن دانه تا نزدیک ترین دانه به خود و به همین ترتیب ادامه میدهیم. به این ترتیب مجموع این مقدار heuristic ما خواهد بود.
این تابع admissible است زیرا در بهترین حالت یعنی عدم برخورد مار به بدن خودش و البته اینکه همه دانه ها امتیاز یک باشند این مقدار خواهد بود، پس همواره:

$$h(n) \leq h^*(n)$$

علاوه بر اینها این تابع consistent هم هست. برای consistency باید رابطه زیر برقرار باشد:

$$h(N) \leq c(N,P) + h(P)$$

در اینجا این رابطه برقرار است. اگر دانه ای نخوریم $c(N,P)$ در بهترین حالت همان مقداری است که از $h(N)$ کم شده و $h(P)$ به دست آمده است پس حالت کوچکتر مساوری برقرار خواهد بود.در صورت برخورد به دانه یک هم همین شرایط دقیقا موجود است و در صورت برخورد به دانه 2 پس از خروج از دانه $h(p)$ افزایش هم یافته پس این شرایط همواره برقرار است.

2- در تابع دوم heuristic میزان مجموع دانه های درون زمین است. این هم admissible است و در اکثر مواقع به جز اینکه دانه ها همگی پشت هم باشند که در این حالت، بخش مساوی شرط برقرار است همواره کوچکتر از هزینه واقعی است.

با توجه به اینکه تابع اول مقدار نزدیک تری به مقدار واقعی را تخمین میزند پس حتما ما را در مسیر بهتری قرار میدهد.

- الگوریتم weighted A*: در این الگوریتم از همان A* با تابع اول استفاده کردیم به جز اینکه :

$$f = g + h * WEIGHT$$

تفاوت الگوریتم ها:
قبل از بررسی دقیق و مقایسه الگوریتم ها، ابتدا جدول اطلاعات را برای هر تست کامل میکنیم:

برای تست 1:

	فاصله جواب	مسیر جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	زمان اجرا
BFS	12	LDUULULLUULL	9524	4531	0.101857
IDS	12	LDUULULLUULL	56640	13457	0.9023
A* 1	12	DLRRDDRRDRDD	1432	1251	0.5
A* 2	12	DLULUULLLLUU	6463	4741	0.11
Weighted weight = 2	14	RUURRDDRDDDDDD	1355	892	0.03
Weighted Weight = 5	14	RUURRDDDRRRRRR	529	381	0.008

برای تست 2:

	فاصله جواب	مسیر جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	زمان اجرا
BFS	15	URDLLUUUULULLLL	108467	48454	1.46
IDS	15	RDLLUUUULULLLL	571852	99912	9.756
A* 1	15	RLLURULULUULLLL	192	191	0.005
A* 2	15	URDLLUULUULLULL	159784	113664	3.38
Weighted weight = 2	15	RULLDLUUUUULLLL	61	46	0.00099
Weighted Weight = 5	15	RULLDLUUUUULLLL	61	46	0.00099

برای تست 3:

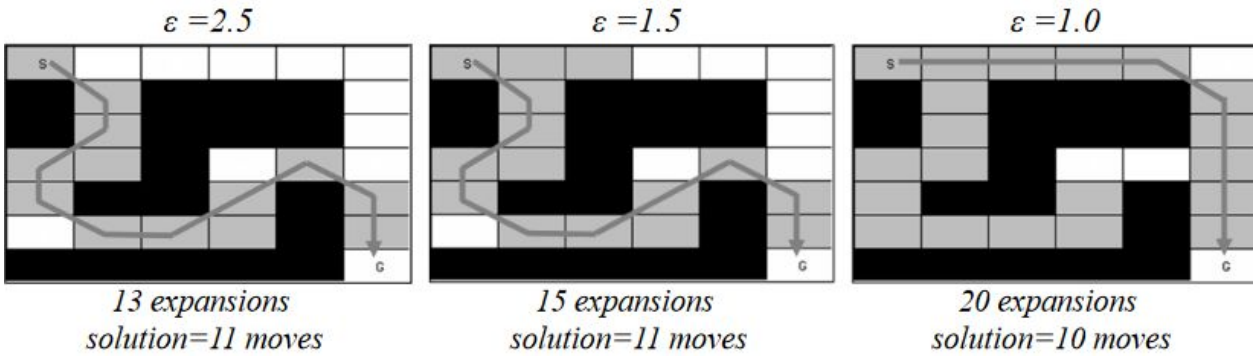
	فاصله جواب	مسیر جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	زمان اجرا
BFS	25	URRDDDRRDRRRDDRRULLDLUULL	445393	198194	5.71
IDS	25	RRDDDRRDRRRDDRRULLDLUULL	3599510	466465	66.935
A* 1	25	URDDDRDRRRDRDRRRULLDLUULL	19430	13510	0.545
A* 2	25	RUDDDRRRDRDRDRURRDLLULLLU	2742249	1533576	68
Weighted weight = 2	26	URRDDDDRRRULDDRDRRURRDLDLU	1976	1351	0.036
Weighted Weight = 5	26	URRDDDRRDDRULDDRRRURRDLDLU	235	172	0.005

با توجه به جداول در مقایسه الگوریتم ها میتوان گفت A* با یک هیوریستیک بسیار مناسب و با تخمین بسیار عالی بسیار وضعیت را خوب می کند و علاوه بر زمان مناسب جواب بهینه را به ما میدهد. Weighted با وزن بالا میتواند در زمان ما را بسیار یاری کند اما در این روش با توجه به وزن بالاتر با اینکه سرعت خوبی می گیریم اما بهینه بودن جواب از دست میرود. در کل A* به طور کلی روش بهتری است اما بسته به نیاز انتخاب روش حل ممکن است متفاوت باشد. در مورد IDS در این مثال ها چون عمق چندان زیادی ندارم خیلی خود را نشان نمی دهد اما به طور کلی استیت های متفاوت کمتری خواهد دید.

اما درباره جواب های متفاوت در Weighted به ازای وزن های متفاوت این است که با توجه به ضریب در رابطه:

$$f = g + h * WEIGHT$$

باید بگوییم به ترتیب استیت ها مانند A* عادی نخواهد بود و برخی استیت هایی که در قیل شانس گسترش داشتند از بین میروند و این باعث میشود استیت های کمتری دیده بشوند و سریعتر به جواب برسیم. وزن بیشتر در اینجا شرایط را بهتر میکند مانند شکل زیر:



همانطور در شکل هم دیده میشود با وزن بیشتر حالات کمتری می بینیم اما ممکن است از جواب بهینه دور شویم.

نتیجه:

میتوان در کل نتیجه گرفت هر روش نسبت به دیگری برتری دارد اما در کل میتوان گفت روش A* روشی بسیار کامل است که در آن هم به سرعت و هم به دقت و بهینه بودن پرداخته شده است.