به نام خدا

پروژه اول مبانی هوش – زهرا رحیمی 9831026

1)پیدا کردن یک نقطه ثابت غذا با استفاده از جستجوی عمق اول (DFS):

سوال) ترتیب کاوش تقریبا همان چیزی بود که حدس می‌زدم اما چون گاهی پکمن وقتی به چند راه می رسد ابتدا مسیر ها را کاوش کرد و اگر به هدف ختم نشد از آن مسیر صرف نظر می‌کند، گاهی طبق فرض من پیش نمی‌رفت. خیر گاهی مربع هایی وجود داشت که به همان دلیلی که گفتم(نرسیدن به هدف) کاوش می‌شدند اما به هدف نمی‌رسیدند.

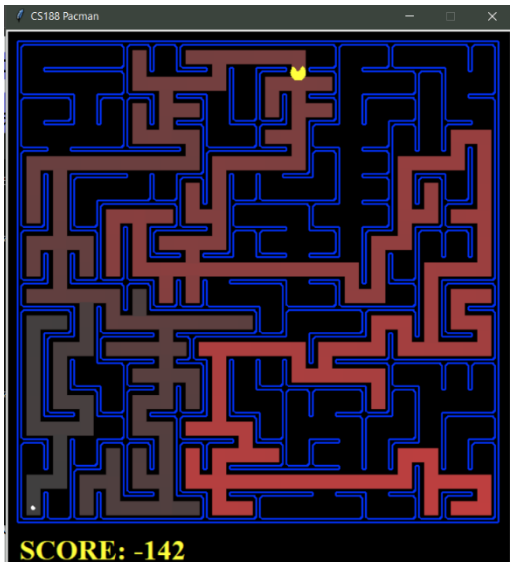
سوال) خیر کمترین هزینه را ندارد. اگر هدف در نقاط سطحی باشد، این الگوریتم مدت ها طول می‌کشد تا به آن برسد چرا که در حال گشتن و جستجوی نقاط عمقی است. در عین حال برای اهداف در عمق های زیاد کارایی خوبی دارد(مانند bigMaze)

2) جستجوی سطح اول یا BFS:

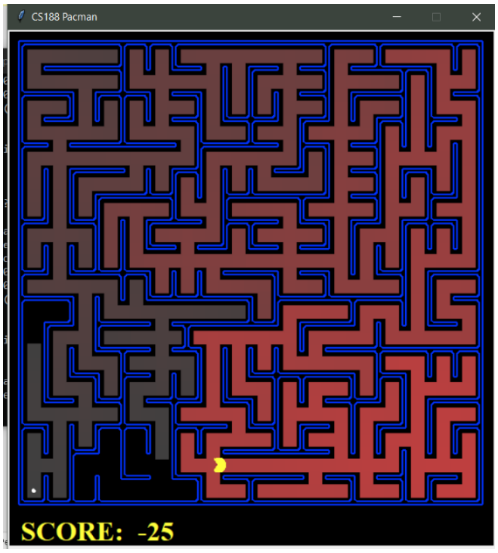
در مسئله bigMaze، از آنجاکه در BFS از صف استفاده می کنیم که به صورت FIFO پیاده سازی شده است، همان گره ای اول کاوش می شود که اول وارد صف شده بود، و در واقع جستجو به صورت لایه لایه انجام می شود. این امر باعث می شود که به goal که در لایه های آخر می باشد، دیرتر برسیم زیرا جستجو سطحی بوده اما هدف در عمق وجود دارد. اما در DFS که از پشته استفاده شده بود که به صورت LIFO، هر گره که اضافه می شود (که در گراف فرزندان دیرتر وارد می شود) نسبت به بقیه گره های داخل پشته اولویت دارند در نتیجه DFS هر بار عمیق تر از بار قبل کاوش می کند در نتیجه به goalای که در عمق های زیاد وجود دارد زودتر خواهد رسید. به علاوه اینکه در DFS، به خاطر استفاده از پشته backtracking داریم اما در BFS صف بدون بازگشت داریم که این در زمان جستجو موثر می‌باشد.

از نتایج هم مشخص است که DFS با 391 جستجوی گره و BFS با 600 جستجو به انتها رسیده اند.

:DFS



:BFS



3) تغییر تابع هزینه:

اگر هزینه همه یال ها 1 باشد الگوریتم به شکل BFS در می آید زیرا priority queue تبدیل به همان queueای می شود که برای BFS استفاده کرده بودیم. اما برای DFS باید کاری کنیم priority queue همواره گره ای را انتخاب کند که جدیدتر آمده باشد، در واقع می توانیم به هر سطح (لایه) یک اولویت نسبت بدهیم به طوری که این اولویت از اولویت لایه قبلی بیشتر و از اولویت لایه بعدی کمتر باشد و اولویت همه گره های هم سطح یکی باشند. به این شکل priority queue وقتی گره فرزند اضافه می کند در نوبت بعد چون اولویت بیشتری از گره والد دارد، آن را کاوش کرده در نتیجه الگوریتم DFS تولید میشود.

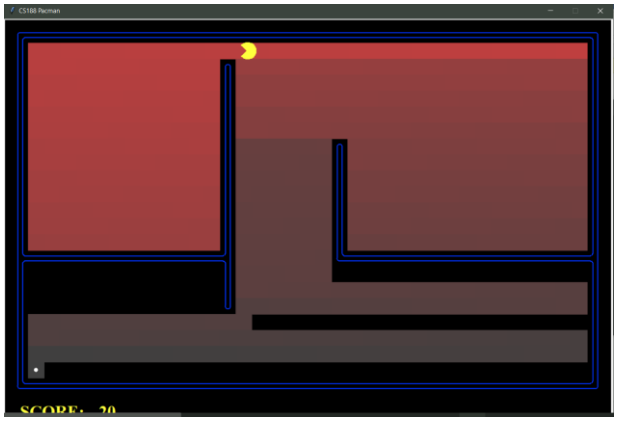
4) جستجوی Astar:

اجرای openMaze برای همه الگوریتم ها:

:DFS

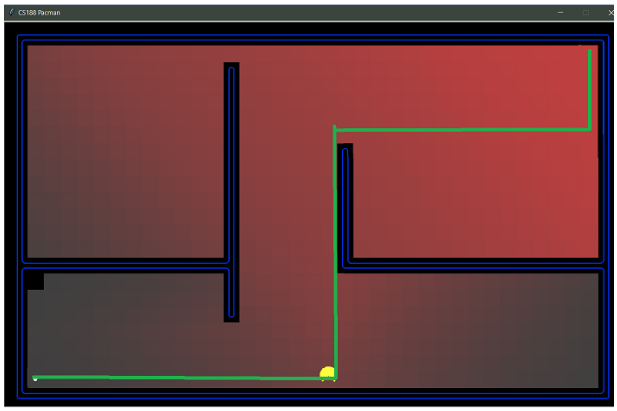
DFS یا خوشحال ترین الگوریتم :)

هر بار تا انتهای مسیر مستقیم می رود و عمق را زیاد می کند به این امید که goal در مکان های عمیق تر باشد. اما چرا در big maze خوب عمل کرد؟ زیرا در آنجا goal در عمق های زیاد بود (و همچنین وجود دیوار ها مسیر خوبی را در پیش روی DFS قرار می دادند) که اتفاقا این عمیق شدن به ما کمک می کرد راه رسیدن به goal سریعتر باشد.



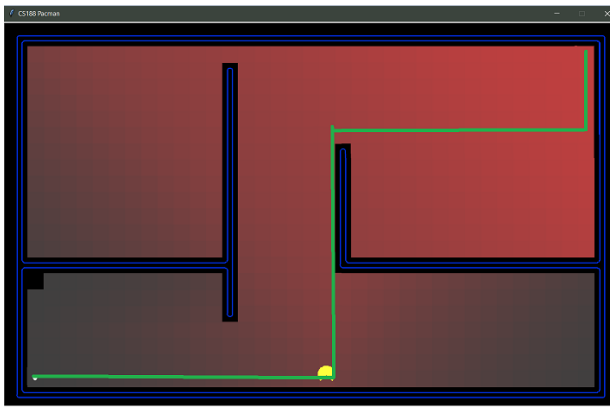
:BFS

زمان جستجوی BFS به دلیل عدم بازگشت و لایه به لایه جستجو کردن بسیار کمتر از DFS می باشد. اما همچنان اشکال بزرگی دارد که به نسبت مسیر نهایی، مکان های زیادی را کاوش می کند.



:UCS

هم از نظر مسیر برگردانده شده هم از نظر مکان های جستجو شده مانند BFS عمل می‌کند چرا که گفتیم اگر هزینه همه گره ها برابر باشد(در اینجا به دلیل کم بودن دیوار ها و نبود غذا و روح هزینه تمام مکان ها تقریبا برابر شده است)، UCS هم مانند BFS یک صف تشکیل می‌دهد که باید همه سطوح را جستجو کند.



A star:

مسیر برگردانده شده همان است اما تعداد مکان‌های جستجو شده با استفاده از هیوریستیک فاصله منهتن، کاهش یافته است.

6) هیوریستیک برای مسئله گوشه‌ها

سوال) برای این مسئله دو هیوریستیک طراحی شد:

هیوریستیک اول: فاصله پکمن تا نزدیکترین غذا را در نظر بگیریم و جمع آن با دورترین فاصله این غذا تا غذاهای دیگر (که در واقع آخرین غذا می باشد) را به عنوان هیوریستیک خروجی بدهیم.

توجه: در همه هیوریستیک ها یک relaxation مبنی بر اینکه هیچ دیواری وجود ندارد در نظر گرفتیم و از فاصله منهتن استفاده می‌کنیم.

برای اثبات قابل قبول بودن می توانیم بدترین حالت را در نظر بگیریم: برای مثال اگر فقط یک گوشه باقی مانده باشد، ما فاصله پکمن تا این گوشه را بدست آورده و با صفر (چون هیچ گوشه دیگری وجود ندارد) جمع می کنیم. پس هزینه هیوریستیک حداکثر به اندازه هزینه پکمن می شود و قابل قبول است

هیوریستیک سازگار است زیرا گره بعدی که ما در این الگوریتم جستجو می کنیم را از دور قبلی به دست آورده ایم که در آنجا اثبات کردیم جمع مبدا تا این گره به علاوه فاصله این گره تا هدف کمترین است. پس گره ای که باز هم در این دور، تولید می شود مقدار کمتری نسبت به مجموع دور قبلی دارد، و از آنجا که ثابت کردیم قابل قبول است و هزینه هیوریستیک کمتر از هزینه واقعی است می توانیم استنتاج کنیم، اختلاف هیوریستیک های این دو هم از هزینه واقعی بین شان، کمتر می باشد.

هیوریستیک دوم: این هیوریستیک شامل دو بخش می باشد:

الف) در یک آرایه گوشه های بازدید نشده را نگه می داریم.

ب) فرض می‌کنیم پکمن ابتدا گوشه هایی را می رود که به موقعیت مکانی او نزدیکتر باشند. در نتیجه فاصله منهتن پکمن تا نزدیکترین گوشه را به‌دست می‌آوریم. سپس برای مرحله بعدی، این گوشه نقش مبدا را می گیرد و باید آن را با بقیه گوشه های پیمایش نشده پیمایش کنیم.

تا وقتی آرایه گوشه های بازدید نشده خالی نشده باشد این دو مرحله را انجام می‌دهیم و مقدار فاصله بدست آمده را جمع زده و بعنوان خروجی تابع می دهیم.

اثبات قابل قبول بودن و سازگاری: از آنجاکه هر سه مرحله قابل قبول است می توانیم تصور کنیم به طور کلی الگوریتم قابل قبول می باشد. علاوه بر این می توان اینگونه توجیه کرد که وقتی ما در هر مرحله کمترین فاصله پکمن را تا گوشه ها بدست می آوریم، هزینه به دست آمده از مجموع این فواصل یقینا کمتر از هزینه واقعی پکمن برای رفتن به چهار گوشه می باشد.

علاوه بر این، ما داریم مسئله ساده شده را در نظر می گیریم ولی هیوریستیک در واقع خود مسئله را حل میکند (هزینه اش تکمیل واقعی نیست در نتیجه timeout نمی‌شود) پس وقتی گوشه ای را از آرایه انتخاب می کنیم که کمترین مقدار با مبدا را دارد، و جمع آن را با بقیه مقادیر کمتر پیدا می کنیم و به عنوان هیوریستیک می‌دهیم، هنگامی که این گره explore و از آرایه خارج می شود، مطمئنا مقدار هیوریستیک جدید مقدار کمتری نسبت به قبلی است که این تفاوت کمتر از اندازه هزینه بین این دو گوشه می باشد. زیرا با ساده سازی های انجام شده(در نظر نگرفتن دیوار و روح) و فاصله منهتن، هزینه واقعی بین دو گوشه کمتر از تفاضل هیوریستیک های آن ها می شود.

هیوریستیک سوم: فاصله تا تمام گوشه های بازدید نشده توسط mazeDistance اندازه گرفته می‌شود و بزرگترین آن به عنوان هیوریستیک برگردانده می شود.

این تابع با استفاده از PositionalSearchProblem و الگوریتم BFS پیاده سازی شده و قابل قبول و سازگار است.

7) خوردن همه غذاها

سوال) برای اینکه یک هیوریستیک مناسب برای این مسئله پیدا کنیم باید به دو موضوع دقت داشته باشیم:

1. قابل قبول باشد: هزینه بازگردانده شده توسط تابع هیوریستیک کمتر از هزینه واقعی رسیدن تا هدف باشد.
2. سازگار باشد: اختلاف هیوریستیک هر دو گره از هزینه واقعی آن دو کمتر باشد.

این هیوریستیک شامل دو بخش می باشد:

الف) در یک آرایه غذا های خورده نشده را نگه می داریم، برای اینکه تخمین مناسبی از غذاهای باقی‌مانده داشته باشیم.

ب) فرض می‌کنیم پکمن ابتدا نقطه غذاهایی را می‌خورد که به موقعیت مکانی او نزدیکتر باشند. در نتیجه فاصله منهتن پکمن تا نزدیکترین غذا را به‌دست می‌آوریم.

تا وقتی آرایه غذاها خالی نشده باشد این دو مرحله را انجام می‌دهیم و مقدار فاصله بدست آمده را جمع زده و بعنوان خروجی تابع می دهیم

اثبات قابل قبول بودن و سازگاری: از آنجاکه هر سه مرحله قابل قبول است می توانیم تصور کنیم به طور کلی الگوریتم قابل قبول می باشد. علاوه بر این می توان اینگونه توجیه کرد که وقتی ما در هر مرحله کمترین فاصله پکمن را تا گوشه ها بدست می آوریم، هزینه به دست آمده از مجموع این فواصل یقینا کمتر از هزینه واقعی پکمن برای رفتن به چهار گوشه می باشد.

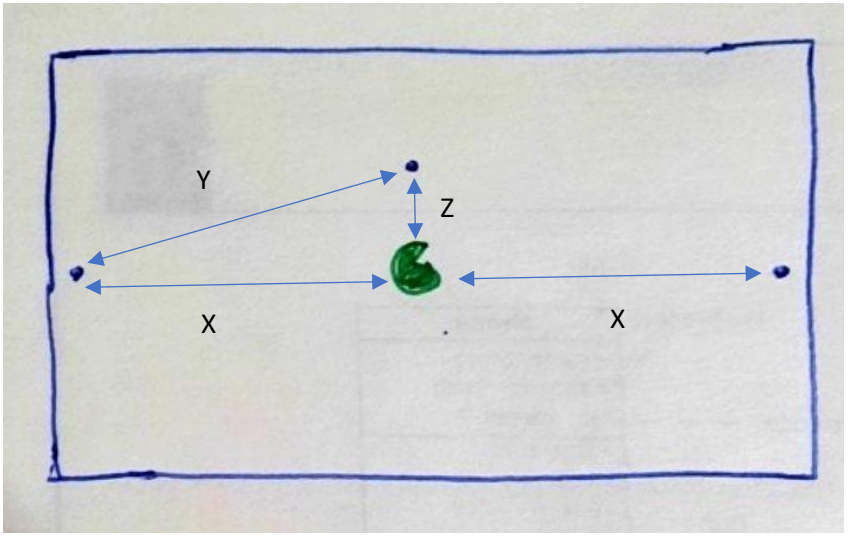
علاوه بر این، ما داریم مسئله ساده شده را در نظر می گیریم ولی هیوریستیک در واقع خود مسئله را حل میکند (هزینه اش تکمیل واقعی نیست در نتیجه timeout نمی‌شود) پس وقتی غذایی را از آرایه انتخاب می کنیم که کمترین مقدار با مبدا را دارد، و جمع آن را با بقیه مقادیر کمتر پیدا می کنیم و به عنوان هیوریستیک می‌دهیم، هنگامی که این گره explore و از آرایه خارج می شود، مطمئنا مقدار هیوریستیک جدید مقدار کمتری نسبت به قبلی است که این تفاوت کمتر از اندازه هزینه بین این دو گوشه می باشد. زیرا با ساده سازی های انجام شده(در نظر نگرفتن دیوار و روح) و فاصله منهتن، هزینه واقعی بین دو غذا کمتر از تفاضل هیوریستیک های آن ها می شود.

هیوریستیک سوم: فاصله تا تمام غذا های خورده نشده توسط mazeDistance اندازه گرفته می‌شود و بزرگترین آن به عنوان هیوریستیک برگردانده می شود.

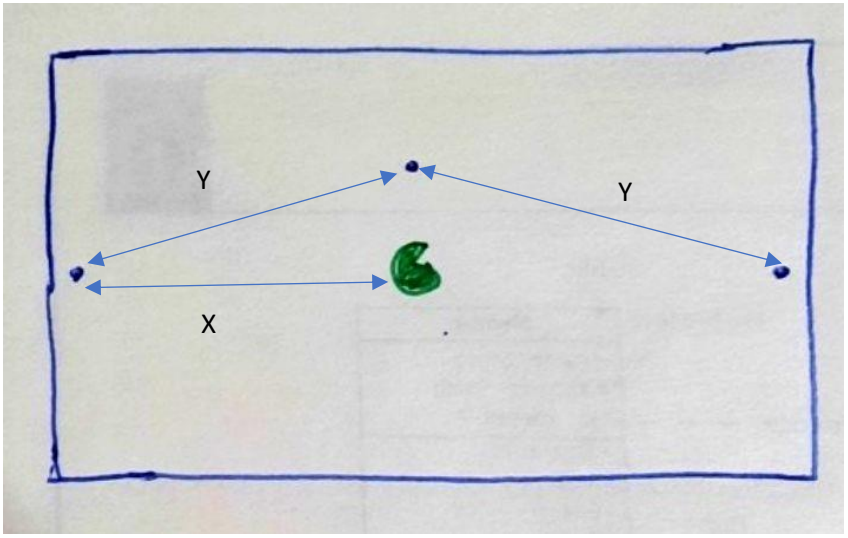
این تابع با استفاده از PositionalSearchProblem و الگوریتم BFS پیاده سازی شده و قابل قبول و سازگار است.

سوال) بسته به هیوریستیک استفاده شده تفاوت ها هم می تواند کم یا زیاد باشد. در اینجا چون تعداد نقاط غذا ها کم بود برای بهتر شدن تخمین هیوریستیک، از همان هیوریستیک گوشه ها استفاده کردم که در این صورت تعداد گره های باز شده حدود 6000 تا بود. اما در کل دو مسئله بسیار شبیه هم بودند که در واقع هر گوشه را یک غذا در نظر می گیریم که باید بازدید / خورده شود.

8) جستجوی نیمه بهینه:



در این فضای حالت پکمن ابتدا به سراغ نقطه بالایی سپس چپ و بعد راست می رود (چپ و راست بستگی به پیاده سازی دارد) در این صورت مسیر به صورت 2X+Y+Z می شود.



اما حالت بهینه وقتی است که پکمن ابتدا به چپ، سپس بالا و سپس راست برود که در این صورت مسیر به اندازه 2Y+X مسافت خواهد داشت. می توان ثابت کرد که X+Z>Y (فاصله منهتن همواره از فاصله وتری بیشتر است) پس رفتن مکرر به نزدیکترین نقطه منجر به یافتن کوتاهترین مسیر برای خوردن تمام نقاط نشد.

(البته در پکمن مسیر اریب وجود ندارد ولی این را هم می توان به عنوان ساده سازی در نظر گرفت که در مسئله تفاوتی ایجاد نمی‌کند)